CHAPTER 3

3. INPUT/OUTPUT TECHNIQUES

3-1 INTRODUCTION

Due to the type of applications in which they are used, the capability to efficiently handle Input/Output (I/O) information is perhaps the most important characteristic of microprocessor systems. The M6800 architecture incorporates supervisory controls and interface devices that permit a wide variety of I/O techniques to be used. This Chapter describes the I/O characteristics of the M6800 system and their use in typical applications.

Most I/O information can be placed in one of two general categories: (1) control and status signals; (2) data that is to be processed by the MPU. Much of the MC6800's flexibility in handling control and status information depends on three system features:

(1) Many of the routine peripheral control tasks can be delegated to the interface adapters.

(2) Because the design of the interface adapters allows the MPU to treat peripherals exactly like other memory locations, the memory reference instructions that operate directly on memory are also used to control peripherals.

(3) While all MPU's must be able to continuously control simple peripherals under program control, in many typical applications, the peripheral information to the MPU is often asynchronous in nature and is best handled on an interrupt basis. The interrupt structure of the MC6800 allows such applications to be processed in an orderly manner, that is, interrupts are handled without disrupting other system tasks in progress.

The currently available interface devices are described in detail in Section 3-4. The various interrupt control techniques are discussed in Sections 3-2 and 3-3.

In the M6800 system, all data movement between family elements (memory and/or peripheral interface adapters) is normally done through the MPU via the Data Bus. This means that the transfers are program controlled, that is, the movement is accomplished by execution of instructions such as Load, Store, Push, Pull, etc. Numerous examples of programmed controlled data transfers are shown throughout this manual. For example, a program for moving 8-bit bytes from a peripheral to memory (at the rate of 43,000 bytes per second) is described in conjunction with the floppy disk application discussed in Section 5-4.

In most system designs, it is possible to "speed up" data movement by surrendering program control and transferring data directly between the other system elements. This bypassing of the MPU, usually called Direct Memory Access (DMA), requires that the MPU be provided with supervisory signals. In addition, external hardware for generating addresses and controlling the transfer must be provided. The MC6800's supervisory control features allow DMA to be accomplished in a variety of ways. The details of implementation depend on the particular system configuration and timing requirements. Several methods and their relative merits are discussed in Section 3-5 of this Chapter.

3-2 MC6800 INTERRUPT SEQUENCES

In a typical application, the peripheral devices may be continuously generating asychronous signals (interrupts) that must be acted on by the MPU. The interrupts may be either requests for service or acknowledgements of services performed earlier by the MPU. The MC6800 MPU provides several methods for automatically responding to such interrupts in an orderly manner.

In the control of interrupts, three general problems must be considered: (1) It is characteristic of most applications that interrupts must be handled without permanently disrupting the task in process when the interrupt occurs. The MC6800 handles this by saving the results of its current activity so that processing can be resumed after the interrupt has been serviced. (2) There must be a method of handling multiple interrupts since several peripherals may be requesting service simultaneously. (3) If some signals are more important to system operation or if certain peripherals require faster servicing than others, there must be a method of prioritizing the interrupts. Techniques for handling each of these problems with the MC6800 will be described in the following paragraphs.

The MPU has three hardware interrupt inputs, Reset ($\overline{RES}$)[1], Non-Maskable Interrupt ($\overline{NMI}$), and Interrupt Request ($\overline{IRQ}$). An interrupt sequence can be initiated by applying a suitable control signal to any of these three inputs or by using the software SWI instruction. The resulting sequence is different for each case.

3-2.1 INTERRUPT REQUEST ($\overline{IRQ}$)

The IRQ input is the mainstay of system interrupt control. Inputs to $\overline{IRQ}$ are normally generated in PIAs and ACIAs but may also come from other user-defined hardware. In either case, the various interrupts may be wire-ORed and applied to the MPU's $\overline{IRQ}$ input. This input is level sensitive; a logic zero causes the MPU to initiate the interrupt sequence[2]. A flow chart of the $\overline{IRQ}$ sequence is shown in Figure 3-2.1-1.

After finishing its current instruction and testing the Interrupt Mask in the Condition Code Register, the MPU stores the contents of its programmable registers in memory locations specified by the Stack Pointer. (Operation of the Stack Pointer is discussed in Section 1-3.4.1.) This stacking process takes seven memory cycles: two each for the Index Register and Program Counter, and one each for Accumulator A, Accumulator B, and the Condition Code Register. The Stack Pointer will have been decremented seven locations and is pointing to the next empty memory location.

The MPU's next step of setting the Interrupt Mask to a logic one is an important aspect of system interrupt control. Setting the mask allows the control program to determine the order in which multiple interrupts will be handled. If it is desirable to recognize another interrupt (of higher priority, for example) before service of the first is complete, the Interrupt Mask can be cleared by a CLI instruction at the beginning of the current service routine. If each interrupt is to be completely serviced before another is recognized, the CLI instruction is omitted and a Return from Interrupt instruction, RTI, placed at the end of the service routine restores the Interrupt Mask status from the stack, thus enabling recognition of subsequent interrupts.

Note that if the former method is selected (immediate enable of further interrupts), the original interrupt service will still eventually be completed. This is due to the fact that the later interrupt also causes the current status to be put on the stack for later completion. This process is general and means that interrupts can be

[1] The bar convention over the symbols is used to indicate an active low signal condition.

[2] $\overline{IRQ}$ is a maskable input. If the Interrupt Mask Bit within the MPU is set, low levels on the $\overline{IRQ}$ line will not be recognized; the MPU will continue current program execution until the mask bit is cleared by encountering the Clear Interrupt (CLI) instruction in the control program, or an RTI is encountered.

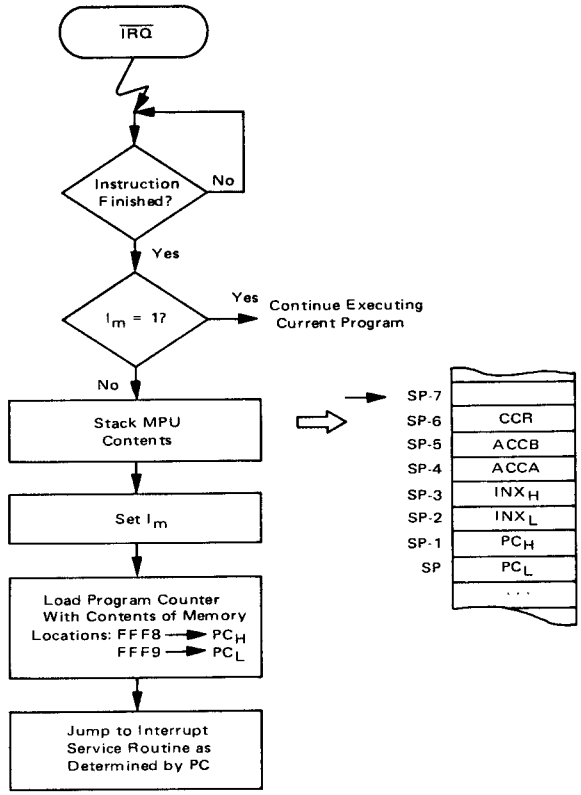


FIGURE 3-2.1-1: Hardware Interrupt Request Sequence

CONTENTS	ADDRESS
$\overline{RES}$ (Low Byte)	FFFF
$\overline{RES}$ (High Byte)	FFFE
$\overline{NMI}$ (Low Byte)	FFFD
$\overline{NMI}$ (High Byte)	FFFC
SWI (Low Byte)	FFFB
SWI (High Byte)	FFFA
$\overline{IRQ}$ (Low Byte)	FFF9
$\overline{IRQ}$ (High Byte)	FFF8

FIGURE 3-2. 1-2: Interrupt Vector, Permanent Memory Assignments

"nested" to any depth required by the system limited only by memory size. The status of the interrupted routines is returned on a Last-In-First-Out (LIFO) basis. That is, the last result to be stacked is the first to be returned to the MPU.

After setting the Interrupt Mask, the MPU next obtains the address of the first interrupt service routine instruction from memory locations permanently assigned to the $\overline{IRQ}$ interrupt input. This is accomplished by loading the Program Counter's high and low bytes from memory locations responding to addresses, FFF8 and FFF9, respectively. The MPU then fetches the first instruction from the location now designated by the Program Counter.

This technique of indirect addressing (also called vectoring) is also used by the other interrupt sequences. The "vectors" are placed in the memory locations corresponding to addresses FFF8 through FFFF as shown in Figure 3-2.1-2 during program development.

The MPU places two of the address bytes in the range FFF8 — FFFF on the Address Bus during interrupt sequences. It should be noted that the vector data is fetched from the memory locations that respond to these addresses even though they may not actually be FFF8 — FFFF. For example, in the memory allocation that was illustrated in Section 1-1.2.1 of Chapter 1, the ROM was assigned the 1024 memory locations between C000 and C3FF (decimal 49152 to 50175) by tying Address Lines A_{15} and A_{14} to the ROM's chip enables:

Address Lines	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ROM Connections	E	E	X	X	X	X	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
			Not Connected													

Notice that if the MPU outputs the address FFFF (all ones) while fetching the vector data for a Reset, it is actually addressing memory location C3FF in the system memory.

The significant point is that the eight locations that *respond* to FFF8 — FFFF must be reserved for the interrupt vectors.

3-2.2 NON-MASKABLE INTERRUPT ($\overline{NMI}$)

As implied by its name, the Non-Maskable Interrupt ($\overline{NMI}$) must be recognized by the MPU as soon as the $\overline{NMI}$ line goes to logic zero. This interrupt is often used as a power-failure sensor or to provide interrupt service to a "hot" peripheral that must be allowed to interrupt.

Except for the fact that it cannot be masked, the $\overline{NMI}$ interrupt sequence is similar to $\overline{IRQ}$ (See Figure 3-2.2-1). After completing its current instruction, the MPU stacks its registers, sets the Interrupt mask and fetches the starting address of the $\overline{NMI}$ interrupt service routine by vectoring to FFFC and FFFD. (See Figure 3-2.1-2).

3-2.3 RESET ($\overline{RES}$)

The Reset interrupt sequence differs from $\overline{NMI}$ and $\overline{IRQ}$ in two respects. When $\overline{RES}$ is low, the MPU places FFFE (the high order byte of the $\overline{RES}$ vector location) on the Address Bus in preparation for executing the $\overline{RES}$ interrupt sequence. It is normally used following power on to reach an initializing program that sets up system starting conditions such as initial value of the Program Counter, Stack Pointer, PIA Modes,

etc. It is also available as a restart method in the event of system lockup or runaway. Because of its use for starting the MPU from a power down state, the $\overline{RES}$ sequence is initiated by a positive going edge. Also, since it is normally used only in a start-up mode, there is no reason to save the MPU contents on the stack. The flow is shown in Figure 3-2.3-1. After setting the Interrupt mask, the MPU loads the Program Counter from the memory locations responding to FFFE and FFFF and then proceeds with the initialization program.

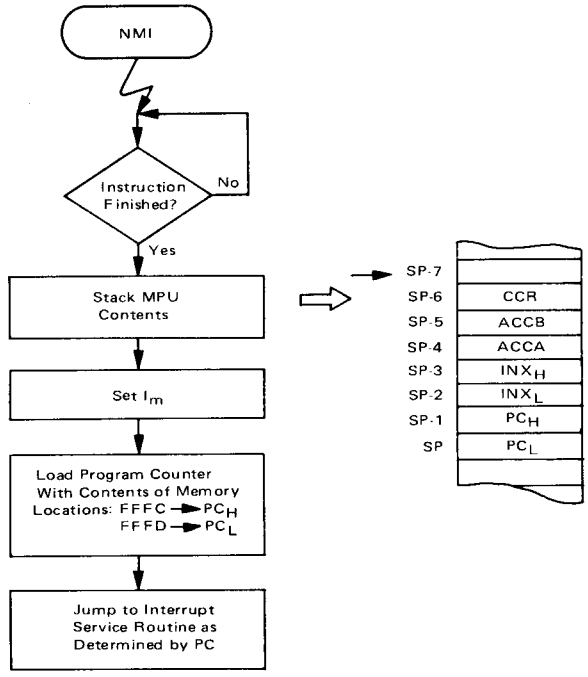


FIGURE 3-2.2.1: Non-Maskable Interrupt Sequence

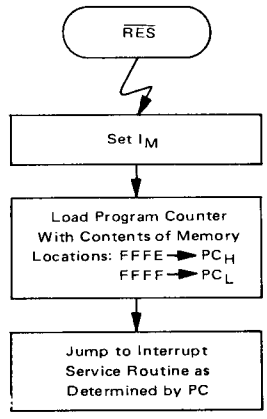


FIGURE 3-2.3-1: Reset Interrupt Sequence

3-2.4 SOFTWARE INTERRUPT (SWI)

The MPU also has a program initiated interrupt mode. Execution of the Software Interrupt (SWI) instruction by the MPU initiates the sequence shown in Figure 3-2.4-1. The sequence is similar to the hardware interrupts except that it is initiated by "software" and the vector is obtained from memory locations responding to FFFA and FFFB.

The Software Interrupt is useful for inserting break-points in the program as an aid in debugging and troubleshooting. In effect, SWI stops the process in place and puts the MPU register contents into memory where they can be examined or displayed.

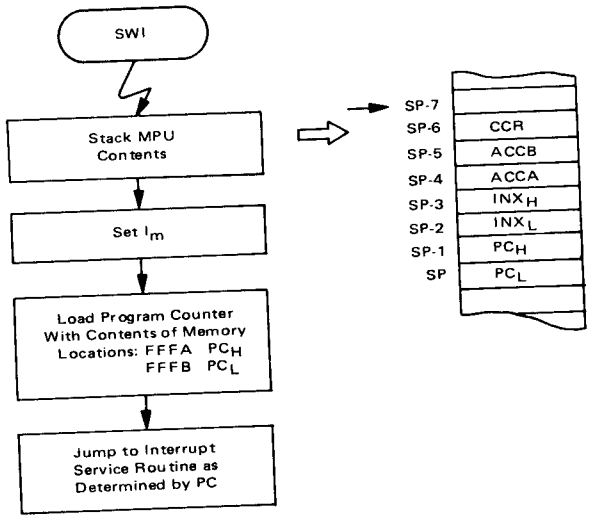


FIGURE 3-2.4-1: Software Interrupt Sequence

3-3 INTERRUPT PRIORITIZING

In the previous section, the various methods available for finding the "beginning" of an interrupt control program were described. If there is only one peripheral capable of requesting service, the source of the interrupt is known and the control program can immediately begin the service routine. More often, several devices are allowed to originate interrupt requests and the first task of the interrupt routine is to identify the source of the interrupt.

There is also the possibility that several peripherals are simultaneously requesting service. In this case, the control program must also decide which interrupt to service first. The $\overline{IRQ}$ interrupt service routine in particular may be complex since most of the I/O interrupts are wire-ORed on this line.

The most common method of handling the multiple and/or simultaneous $\overline{IRQ}$ interrupts is to begin the service routine by "polling" the peripherals to see which one generated the request. If the interrupts are generated by peripheral signals coming in through a PIA or an ACIA, the polling procedure is very simple. In addition to causing $\overline{IRQ}$ to go low, the interrupting signal also sets a flag bit in the PIA's or ACIA's internal registers. Since these registers represent memory locations to the MPU, the polling consists of nothing more than stepping through the locations and testing the flag bits[3].

Establishing the priority of simultaneous interrupts can be handled in either of two ways. The simplest is to establish priority by the order in which the PIAs and ACIAs are polled. That is, the first I/O flag encountered gets the service, so higher priority devices are polled first. The second method first finds all the interrupt flags and then uses a special program to select the one having highest priority. This method permits a more sophisticated approach in that the priority can be modified by the control program. For example, it might be desirable to select the lower priority of two simultaneous requests if the lower priority has not been serviced for some specified period of time.

Software techniques can, in theory, handle any number of devices to any sophistication level of prioritizing. In practice, if there are many sources of interrupt requests, the time required to find the appropriate interrupt can exceed the time available to do so. In this situation, external prioritizing hardware can be used to speed up the operation.

One method for implementing hardware prioritized interrupts is shown in block diagram form in Figure 3-3-1. With this technique, each interrupting device is assigned its own address vector which is stored in ROM memory similarly to the $\overline{RES}$, SWI, $\overline{IRQ}$, and $\overline{NMI}$ vectors. An external hardware priority encoder selects the interrupt to be recognized and directs the MPU to the proper locations in memory for obtaining the vectors.

Operation of the MPU itself is unchanged; after recognizing an $\overline{IRQ}$, the MPU still outputs addresses FFF8 and FFF9 as before. However, some of the address lines are no longer tied directly to memory but go instead to a 1-of-2 Data Selector. The other set of inputs to the Data Selector are generated by a Priority Encoder that outputs a binary number corresponding to the highest priority interrupt signal present at the time the interrupt was recognized by the MPU.

Detection of the FFF8 and FFF9 addresses by the Address Bus monitoring circuitry then causes the outputs of the priority encoder to be substituted for part of the normal address. Hence, even though the MPU outputs FFF8 and FFF9, other locations in ROM are read by the MPU. Suitable vectors for sending the MPU directly to the appropriate service routine are stored in these locations. Specific circuits for implementing this prioritizing method are described in Section 4-2.1.

[3]See Section 5-4 for a specific example of software polling.

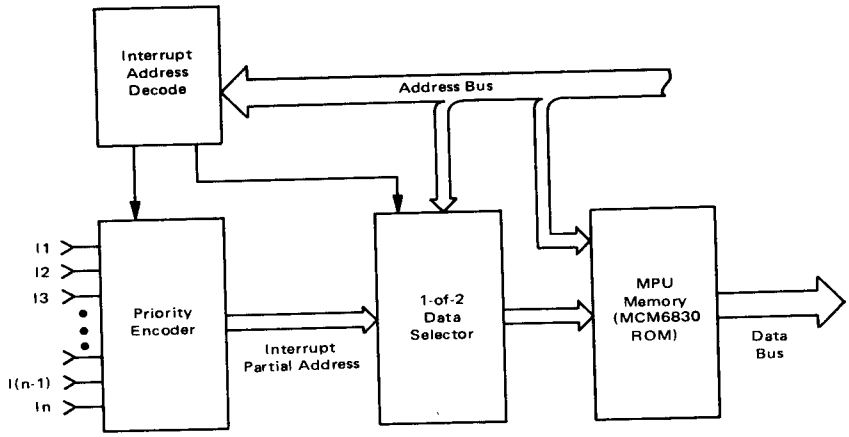


FIGURE 3-3-1: Hardware Interrupt Prioritizing Block Diagram

3-4 PROGRAM CONTROLLED DATA TRANSFERS

3-4.1 MC6820 PERIPHERAL INTERFACE ADAPTER

3-4.1.1 Input/Output Configuration:

The MC6820 Peripheral Interface Adapter (PIA) provides a flexible method of connecting byte-oriented peripherals to the MPU. The PIA, while relatively complex itself, permits the MPU to handle a wide variety of equipment types with minimum additional logic and simple programming. An Input/Output Diagram of the MC6820 is shown in Figure 3-4.1.1-1.

Data flows between the MPU and the PIA on the System Data Bus via eight bi-directional data lines, D0 through D7. The direction of data flow is controlled by the MPU via the Read/Write input to the PIA.

The "MPU side" of the PIA also includes three chip select lines, CS0, CS1, and CS2, for selecting a particular PIA. Two addressing inputs, RS0, and RS1, are used in conjunction with a control bit within the PIA for selecting specific registers in the PIA. The MPU can read or write into the PIA's internal registers by addressing the PIA via the system Address Bus using these five input lines and the R/W signal. From the MPU's point of view, each PIA is simply four memory locations that are treated in the same manner as any other read/write memory.

The MPU also provides a timing signal to the PIA via the Enable input. The Enable (E) pulse is used to condition the PIA's internal interrupt control circuitry and for the timing of peripheral control signals. Since all data transfers take place during the $\phi 2$ portion of the clock cycle, the Enable pulse is normally $\phi 2$[4].

The "Peripheral side" of the PIA includes two 8-bit bi-directional data buses (PA0-PA7 and PB0-PB7), and four interrupt/control lines, CA1, CA2, CB1, and CB2. All of the lines on the "Peripheral Side" of the PIA are compatible with standard TTL logic. In addition, all lines serving as outputs on the "B" side of each PIA (PB0-PB7, CB1, CB2) will supply up to one milliamp of drive current at 1.5 volts.

[4]See Section 4-1.3 for exceptions required in some applications.

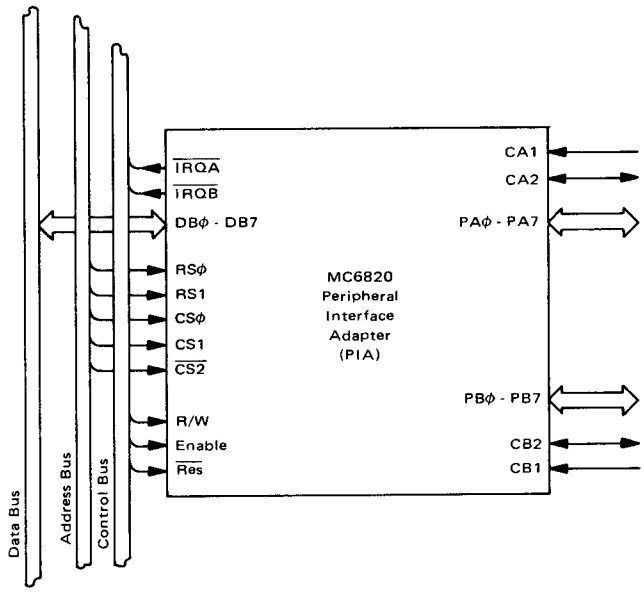


FIGURE 3-4.1.1-1: MC6820 PIA I/O Diagram

3-4.1.2 **Internal Organization:**

An expanded Block Diagram of the PIA is shown in Figure 3-4.1.2-1. Internally, the PIA is divided into two symmetrical independent register configurations. Each half has three main features: an Output Register, a Control Register, and a Data Direction Register. It is these registers that the MPU treats as memory locations, i.e., they can be either read from or written into. The Output and Data Direction Registers on each side represent a single memory location to the MPU. Selection between them is internal to the PIA and is determined by a bit in their Control Register.

The Data Direction Registers (DDR) are used to establish each individual peripheral bus line as either an input or an output. This is accomplished by having the MPU write "ones" or "zeros" into the eight bit positions of the DDR. Zeros or ones cause the corresponding peripheral data lines to function as inputs or outputs, respectively.

The Output Registers, ORA and ORB, when addressed, store the data present on the MPU Data Bus during an MPU write operation[5]. This data will also appear on those peripheral lines that have been

[5]As used here, an "MPU Write" operation refers to the execution of the "Store" instruction, i.e., writing into Output Register A is equivalent to execution of STAA PIAORA by the MPU. Similarly, an "MPU Read" operation is equivalent to execution of the "Load" instruction: LDAA PIAORA.

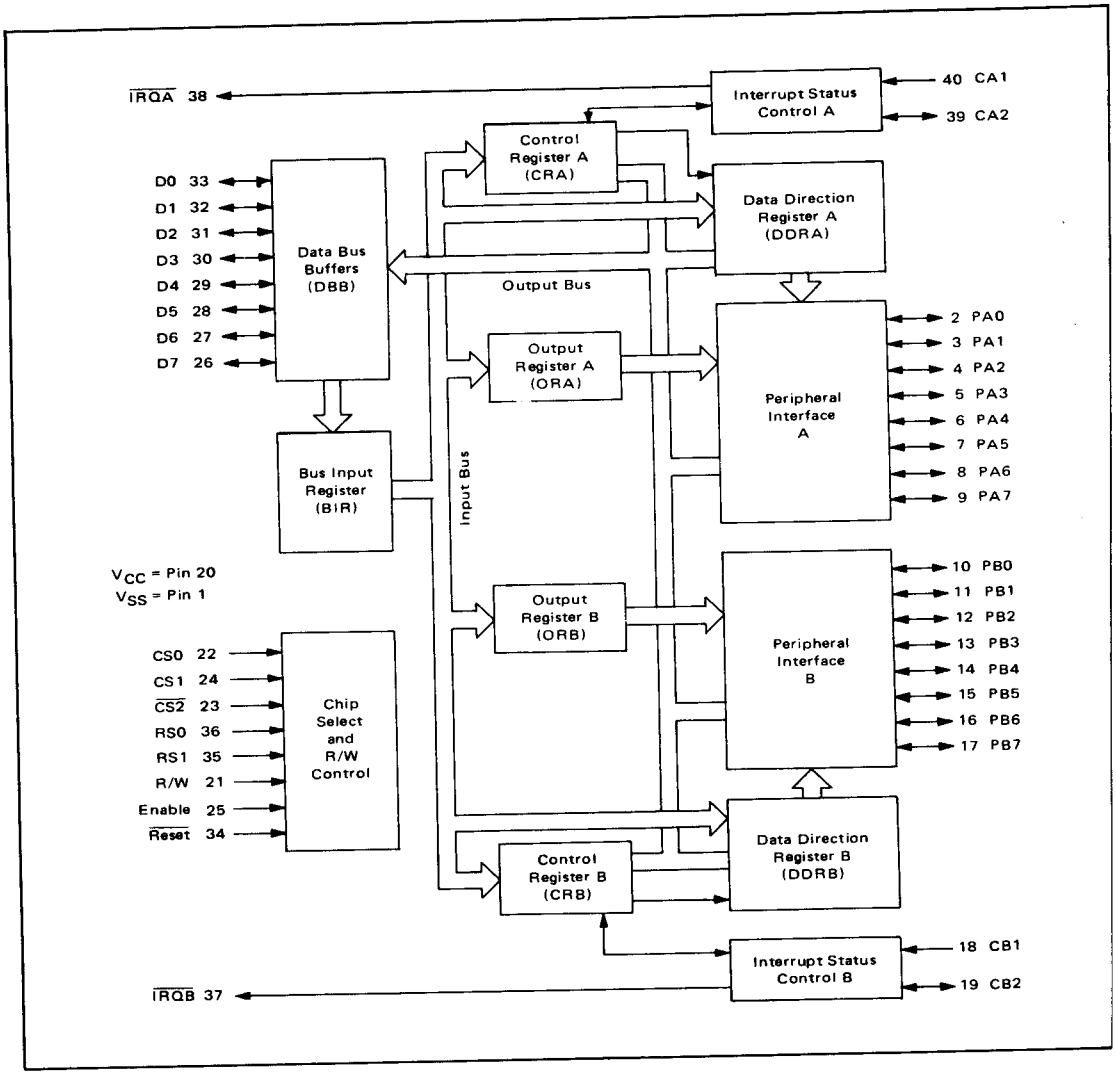


FIGURE 3-4.1.2-1: MC6820 PIA – Block Diagram

programmed as outputs. If a peripheral line has been programmed as an input, the corresponding bit position of the Output Register can still be written into by the MPU, however, the data will be influenced by the external signal applied on that peripheral data line.

During an MPU Read operation, the data present on peripheral lines programmed as inputs is transferred directly to the system Data Bus. Due to differing circuitry, the results of reading positions programmed as outputs differ slightly between sides A and B of the PIA. On the B side, there is three-state buffering between Output Register B and the peripheral lines such that the MPU will read the current contents of ORB for those bit positions programmed as outputs. (See Figure 3-4.1.2-2.) During an MPU Read of the A side, the data present on the Peripheral lines will effect the MPU Data Bus regardless of whether the lines are programmed as outputs or inputs. The bit positions in ORA designated as outputs will be read correctly only if the external loading on the Peripheral lines is within the specification for one TTL load. That is, a logic one level could be read as a logic zero if excessive loading reduced the voltage below 2.0 volts.

The two Control Registers, CRA and CRB, allow the MPU to establish and control the operating modes of the peripheral control lines, CA1, CA2, CB1, and CB2. It is by means of these four lines that control information is passed back and forth between the MPU and peripheral devices. The control word format and a summary of its features is shown in Figure 3-4.1.2-3.

The Data Direction Register access bit (b_2 = DDR Access) is used in conjunction with the register select lines to select between internal registers. For a given register select combination, the status of the DDR bit determines whether the Data Direction Register (b_2 of DDR = 0) or the Output Register (b_2 of DDR = 1) is addressed by the MPU.

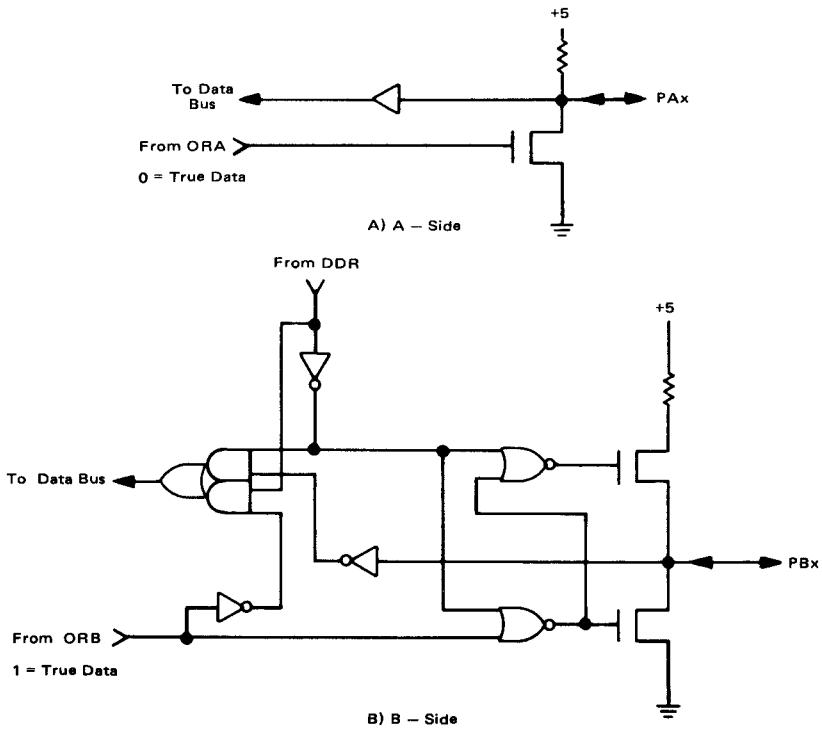


FIGURE 3-4.1.2-2: PIA Output Circuit Configurations

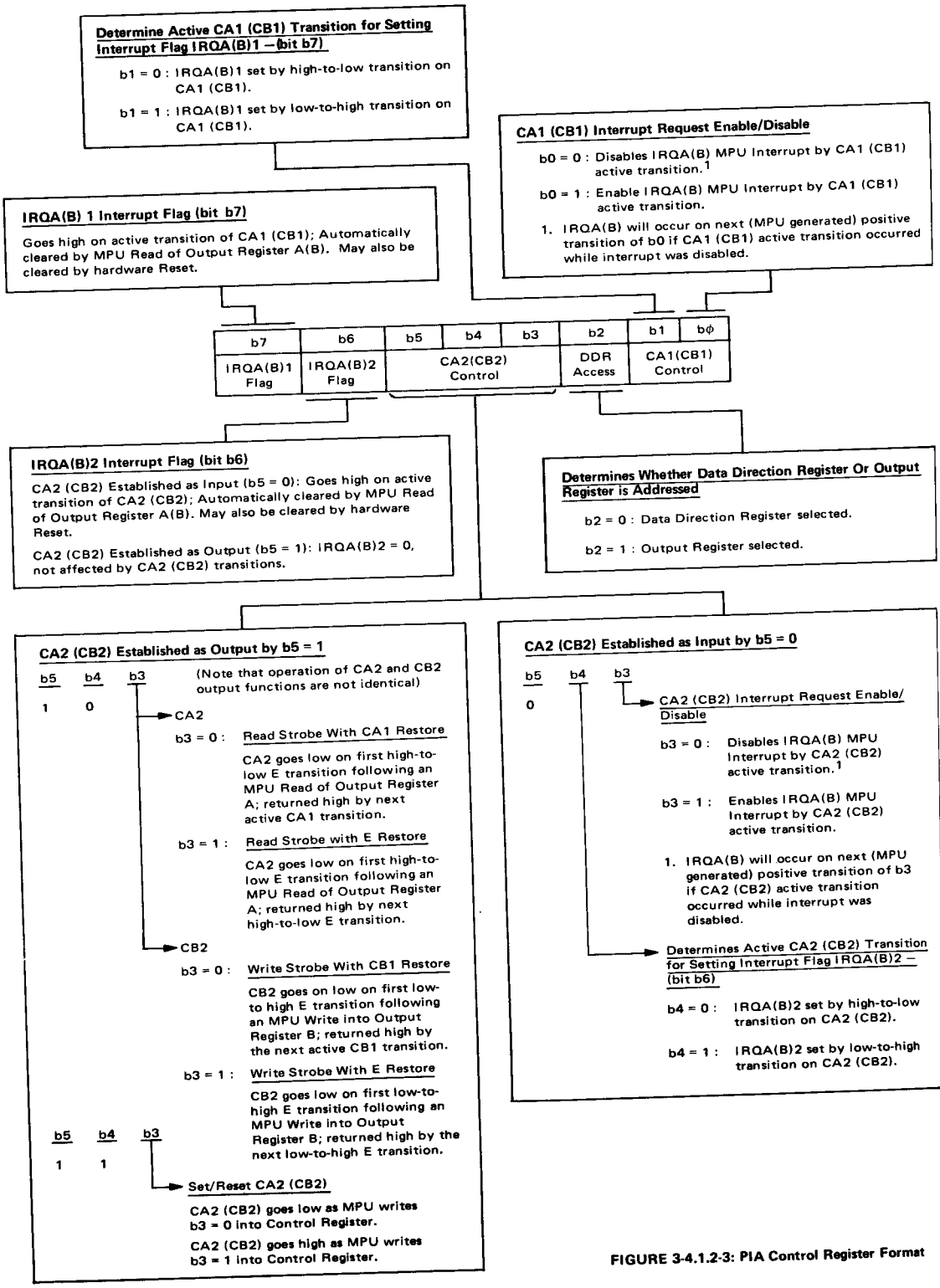


FIGURE 3-4.1.2-3: PIA Control Register Format

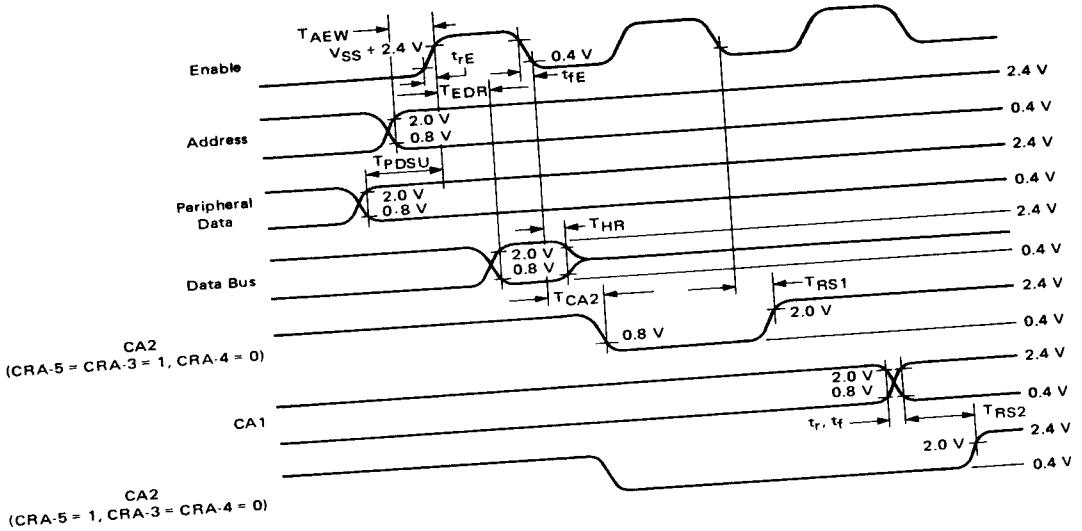
Each Control Register has two interrupt request flags, b_7 = IRQA(B)1 and b_6 = IRQA(B)2; they are set by transitions on the CA1(CB1) and CA2(CB2) control lines and can be read by an MPU read Control Register operation. The status of the interrupt flags cannot be altered by an MPU write instruction, that is, IRQA(B)1 and IRQA(B)2 are Read Only with respect to the MPU. They are indirectly reset to zero each time the MPU reads the corresponding Output Register or can be cleared with the hardware $\overline{\text{Reset}}$.

Bits b_0 and b_1 of the Control Registers determine the CA1(CB1) operating mode. A "one" written into b_1 by the MPU will cause subsequent positive-going transitions of the CA1(CB1) input to set IRQA(B)1; if $b_1 = 0$, negative-going transitions on CA1(CB1) cause IRQA(B)1 to set. If $b_0 = 1$ when the IRQA(B)1 flag goes high, the PIA's external interrupt request line, IRQA(B), immediately goes low, providing a hardware interrupt signal to the MPU. The external interrupt is disabled if $b_0 = 0$ when the internal interrupt is set by CA1(CB1). If b_0 is later set by an MPU Write Control Register operation, the disable is immediately released and a pending external interrupt request will occur.

When $b_5 = 0$, b_3 and b_4 of the Control Register perform similarly to b_0 and b_1 , controlling the $\overline{\text{IRQ}}$A(B)2 interrupt via the CA2(CB2) input. The IRQA(B) interrupt terminal, when enabled, responds to either IRQA(B)1 or IRQA(B)2.

If $b_5 = 1$, CA2(CB2) acts as an output and will function in one of three modes. If b_4 is also equal to one, CA2(CB2) serves as a program-controlled set/reset output to the peripheral and follows b_3 as it is changed by MPU Write Control Register operations. If $b_4 = 0$ when $b_5 = 1$, CA2(CB2) can be used in either a pulse-strobed or handshake mode. Operation of the two sections differ slightly for these two operating modes. In the handshake mode ($b_3 = 0$) CA2 is taken low by the negative transition of the MPU Enable Pulse following an MPU Read Output Register operation and returns high when IRQA1 is next set by CA1. This, in effect, tells the peripheral it has been read and allows it to acknowledge via CA1. The "B" Side operation is similar except that CB2 is taken low following an MPU Write Output Register operation and returned high by the next CB1 transition; this tells the peripheral it has been written into and allows it to respond via CB1.

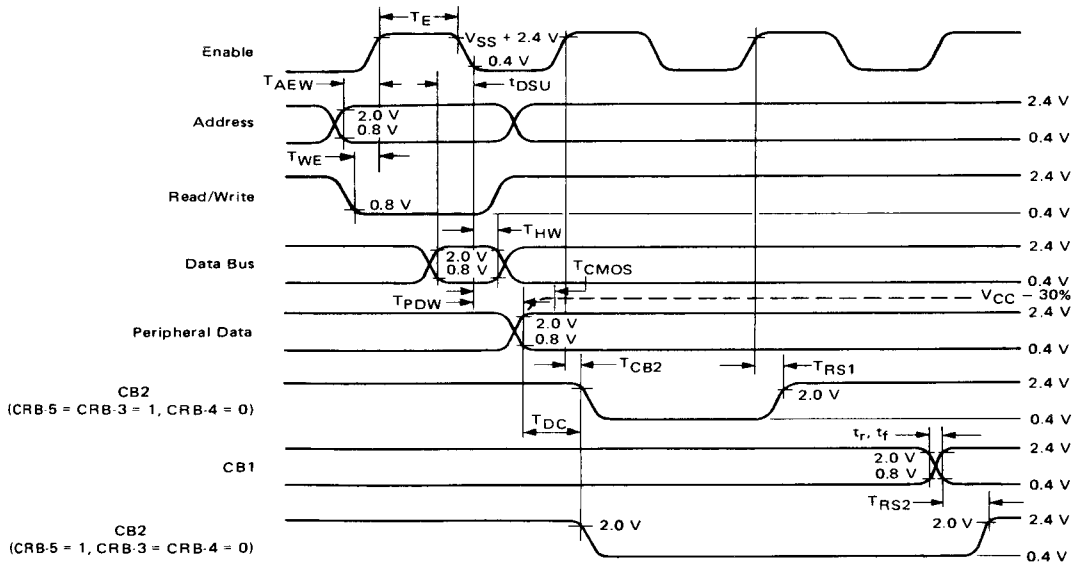
In the pulse-strobed mode ($b_3 = 1$), CA2 is again set low by a Read Output Register command, but is now returned high by the negative transition of the next MPU originated Enable Pulse. CB2 operation is similar except that an MPU Write Operation initiates the pulse. Relative timing waveforms for the strobe control modes are shown in Figures 3-4.1.2-4 and 3-4.1.2-5. The use of A side for Read and B side for Write in those figures is not meant to imply that the A and B sides must be used only for peripheral data in and out, respectively. However, the strobe modes are implemented only as shown, i.e., a strobe is not generated by an A side Write or a B side Read. Strobes can be generated for these cases by including "dummy" instructions in the program. For example, an A side Write instruction can be followed immediately by an A side dummy Read to generate the strobe. Similarly, a B side Read can be followed by a dummy Write.



Loading = 30 pF and one TTL load for PA0-PA7, PB0-PB7, CA2, CB2
= 130 pF and one TTL load for D0-D7, $\overline{IRQA}$, $\overline{IRQB}$)

Characteristic	Symbol	Min	Typ	Max	Unit
Delay Time, Address valid to Enable positive transition	T_{AEW}	180	–	–	ns
Delay Time, Enable positive transition to Data valid on bus	T_{EDR}	–	–	395	ns
Peripheral Data Setup Time	T_{PDSU}	300	–	–	ns
Data Bus Hold Time	T_{HR}	10	–	–	ns
Delay Time, Enable negative transition to CA2 negative transition	T_{CA2}	–	–	1.0	µs
Delay Time, Enable negative transition to CA2 positive transition	T_{RS1}	–	–	1.0	µs
Rise and Fall Time for CA1 and CA2 input signals	t_r, t_f	–	–	1.0	µs
Delay Time from CA1 active transition to CA2 positive transition	T_{RS2}	–	–	2.0	µs
Rise and Fall Time for Enable input	t_{rE}, t_{fE}	–	–	25	ns

FIGURE 3-4.1.2-4: Read Timing Characteristics



Characteristic	Symbol	Min	Typ	Max	Unit
Enable Pulse Width	T_E	0.470	–	25	μs
Delay Time, Address valid to Enable positive transition	T_{AEW}	180	–	–	ns
Delay Time, Data valid to Enable negative transition	T_{DSU}	300	–	–	ns
Delay Time, Read/Write negative transition to Enable positive transition	T_{WE}	130	–	–	ns
Data Bus Hold Time	T_{HW}	10	–	–	ns
Delay Time, Enable negative transition to Peripheral Data valid	T_{PDW}	–	–	1.0	μs
Delay Time, Enable negative transition to Peripheral Data Valid, CMOS (V_{CC} – 30%) PA0-PA7, CA2	T_{CMOS}	–	–	2.0	μs
Delay Time, Enable positive transition to CB2 negative transition	T_{CB2}	–	–	1.0	μs
Delay Time, Peripheral Data valid to CB2 negative transition	T_{DC}	0	–	1.5	μs
Delay Time, Enable positive transition to CB2 positive transition	T_{RS1}	–	–	1.0	μs
Rise and Fall Time for CB1 and CB2 input signals	t_r, t_f	–	–	1.0	μs
Delay Time, CB1 active transition to CB2 positive transition	T_{RS2}	–	–	2.0	μs

FIGURE 3-4.1.2-5: Write Timing Characteristics

3-4.1.3 Addressing and Initialization:

Chapters 6 and 7 of this manual include numerous examples of PIA addressing and initialization, however, some basic considerations are discussed in the following paragraphs. As indicated in Section 3-4.1.1, the MPU addresses the PIA via the five chip select and register select inputs and bit 2 of the Control Registers. The correspondence between internal registers and the address inputs is shown in Figure 3-4.1.3-1.

$\overline{CS2}$	CS1	CSϕ	RS1	RSϕ	b2	
ϕ	1	1	ϕ	ϕ	ϕ	Data Direction Register A (PIADRA)
ϕ	1	1	ϕ	ϕ	1	Output Register A (PIAORA)
ϕ	1	1	ϕ	1	X	Control Register A (PIACRA)
ϕ	1	1	1	ϕ	ϕ	Data Direction Register B (PIADRB)
ϕ	1	1	1	ϕ	1	Output Register B (PIAORB)
ϕ	1	1	1	1	X	Control Register B (PIACRB)
X	X	ϕ	X	X	X	PIA Not Selected
X	ϕ	X	X	X	X	PIA Not Selected
1	X	X	X	X	X	PIA Not Selected

X = Doesn't Matter

FIGURE 3-4.1.3-1: PIA Register Addressing

Addressing a PIA can be illustrated in conjunction with the simple system configuration shown in Figure 3-4.1.3-2[6]. The method shown is typical for assigning mutually exclusive memory addresses to the family devices without using additional address decode logic. The connections shown in Figure 3-4.1.3-2 assign memory addresses as follows:

RAM	0000 – 007F
PIA	4004 – 4007
ACIA	4008 – 4009
ROM	C000 – C3FF

(Hexadecimal notation)

In most cases, the desired I/O configuration and Control Register modes are established as part of an initialization sequence. The steps involved depend on the particular application but can be clarified by means of a specific example.

Assume that a PIA is to be used as the interface between two peripherals. When interrupted by a positive transition on a control line, the MPU is to fetch 8 bits of data from Peripheral #1 and then send an acknowledgement pulse. The MPU must be able to transfer a byte of data to Peripheral #2 and receive acknowledgement that it was accepted. Peripheral #2 must be provided with a control signal indicating that there is data ready for it.

A suitable hardware configuration is shown in Figure 3-4.1.3-3. Peripheral Lines PA0-PA7 are assigned to "read" Peripheral #1 and, hence, must be established as inputs. CA1 provides the interrupt input and must be conditioned to recognize incoming positive transitions. CA2 will be used to signal that data has been read, hence, it must be established as an output using the pulse strobe mode, i.e., reading PIAORA[7] will automatically transmit a pulse to the peripheral.

Peripheral Lines PB0-PB7 are assigned for transmitting data to Peripheral #2 and, hence, must be established as outputs. CB2 will be used as an output for signalling that there is data ready. CB1 will be

[6] Figure 3-4.1.3-2 is identical to Figure 1-1.2-1 and is discussed in Section 1-1.2 of Chapter 1.

[7] In order to use symbolic labels instead of absolute addresses in the initialization program, the labels introduced in Figure 3-4.1.3-1 will be used to refer to PIA registers.

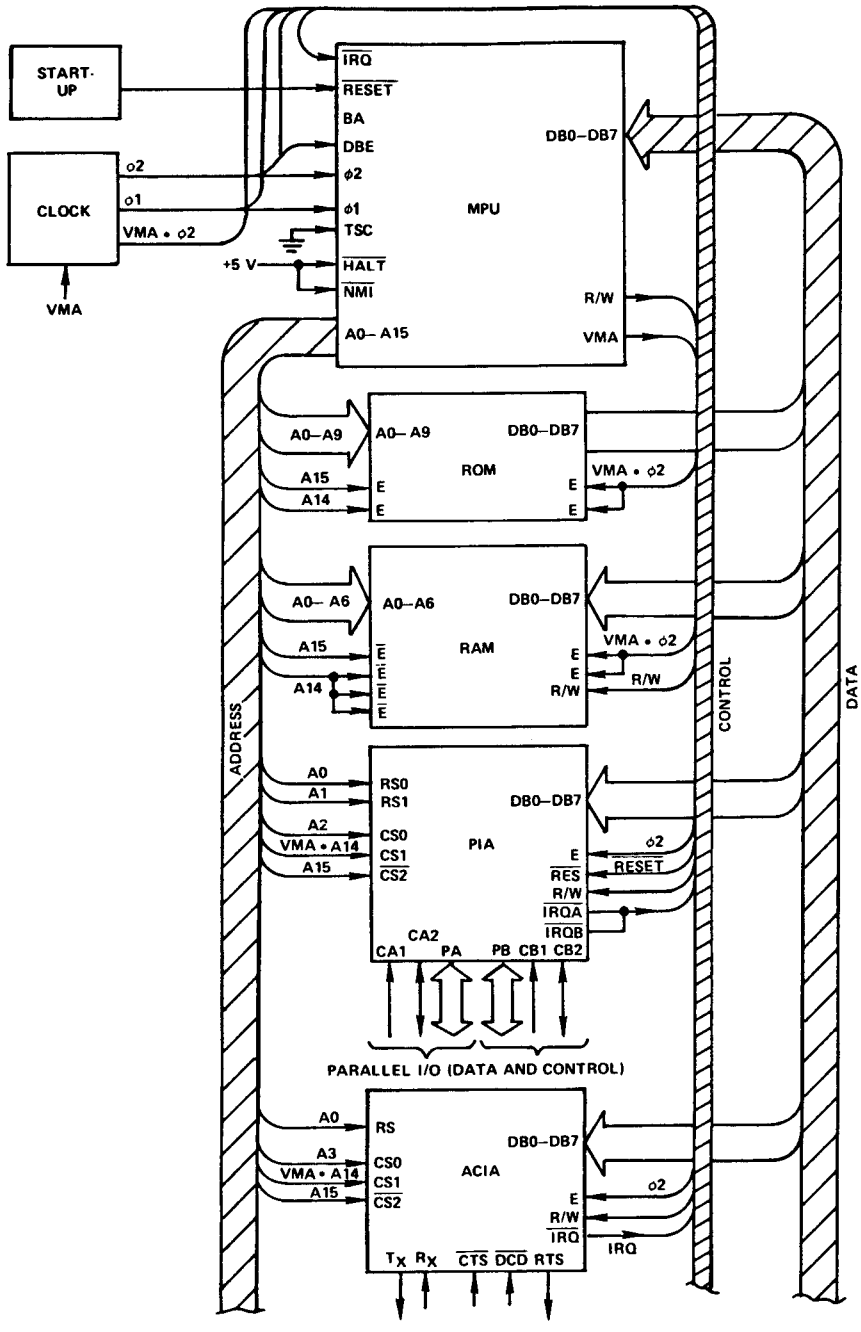


FIGURE 3-4.1.3-2: Family Addressing

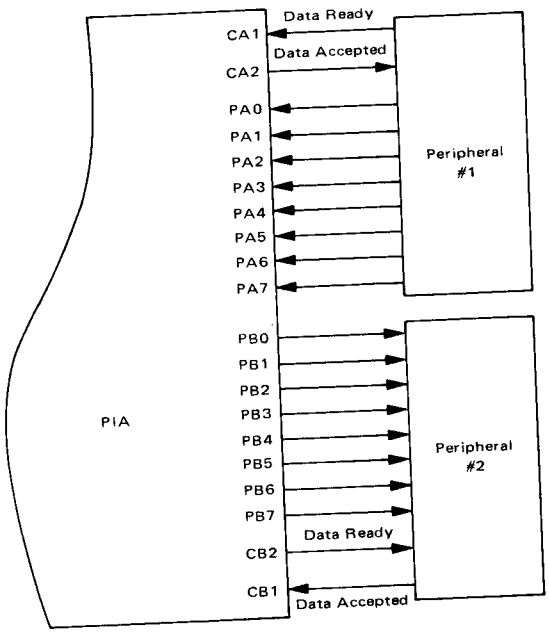


FIGURE 3-4.1.3-3: Typical I/O Configuration

conditioned to accept a negative transition acknowledgement signal from Peripheral #2. CB2 is to be restored by that transition.

If it is known that a hardware system $\overline{\text{Reset}}$ is to be applied prior to initializing, all PIA register bits will be zero initially and the following sequence can be used:

```
10  LDAA   #$2F      SELECT ORA; SET MODE CONTROL
20  STAA   PIACRA        FOR "A" SIDE
30  COM    PIADRB    ESTABLISH PB0-PB7 AS OUTPUTS
40  LDAA   #$24      SELECT ORB; SET MODE CONTROL
50  STAA   PIACRB        FOR "B" SIDE
```

The constant[8] $2F = 00101111 loaded into the A Control Register by Instruction 20 has the following effect: b0 = 1 enables a CA1 interrupt; b1 = 1 selects positive transition for interrupt recognition; b2 = 1 selects ORA (the initial zeros in DDRA establish PA0-PA7 as inputs); b3 = 1, b4 = 0 selects read strobe with E restore; b5 = 1 establishes CA2 as an output; b6 and b7 are don't cares since MPU cannot write into those two positions:

b7	b6	b5	b4	b3	b2	b1	n0	
0	0	1	0	1	1	1	1	= 2F (Hex)

Instruction 30 writes "ones" into the B Data Direction Register, thus establishing PB0-PB7 as outputs. The constant loaded into the B Control Register by instruction 50 has the following effect: b0 = 0 disables $\overline{\text{IRQB}}$ interrupt by CB1 transition (it is assumed that the MPU will read flag bit b7 to check for acknowledgement rather than allowing an interrupt); b1 = 0 selects recognition of negative transition on CB1 for setting flag bit 7; b2 = 1 selects ORB; b3 = 0, b4 = 1 selects Write strobe with CB1 restore; b5 = 1 establishes CB2 as an output; b6 and b7 are don't cares:

b7	b6	b5	b4	b3	b2	b1	b0	
0	0	1	0	0	1	0	0	= 24 (Hex)

If there is no assurance that the PIA internal register bit positions are initially zero prior to initialization, the following sequence can be used:

```
10  CLRA             SELECT
20  STAA   PIACRA        DATA DIRECTION REGISTER A
30  STAA   PIACRB        AND DATA DIRECTION REGISTER B.
40  STAA   PIADRA    ESTABLISH PA0-PA7 AS INPUTS.
50  LDAA   #$2F      SELECT ORA; SET MODE
60  STAA   PIACRA        CONTROL FOR "A" SIDE.
70  LDAA   #$FF      ESTABLISH
80  STAA   PIADRB        PB0-PB7 AS OUTPUTS.
90  LDAA   #$24      SELECT ORB; SET MODE
100 STAA   PIACRB        CONTROL FOR "B" SIDE.
```

Note that if the initialization sequence is started from a known hardware clear only half as many instructions are required.

[8]Refer to Figure 3-4.1.2-3 for derivation of the Control Register words.

3-4.1.4 System Considerations:

The information provided in the preceding paragraphs has been limited to only the more obvious characteristics of the PIA. The features described greatly simplify I/O processing, as will be seen in the examples of later chapters. There are several general techniques worth considering as a system is configured.

The fact that the PIA registers are treated as memory combined with the fact that many of the MPU's instructions (CLR, ASL, COM, TST, etc) operate directly on memory makes possible a variety of I/O techniques. This characteristic should be given careful attention when hardware/software tradeoffs are being considered.

The flexibility inherent in being able to change the I/O direction of individual peripheral lines under program control was not adequately stressed in the initialization discussion. A detailed example making use of this feature to decode a switch matrix is included in Section 5-1.1.1.

Only a simple case of address assignment was considered. Other approaches may lead to a more efficient system. As an example, consider the memory allocation that results from applying A0, and A1 of the address bus to RS0 and RS1, respectively:

RS1 (A1)	RS0 (A0)	
0	0	PIAORA
0	1	PIACRA
1	0	PIAORB
1	1	PIACRB

Here the registers alternate between output and Control[9] Registers. If A0 is connected to RS1 and A1 to RS0, the following result is obtained:

RS1 (A0)	RS0 (A1)	
0	0	PIAORA
1	0	PIAORB
0	1	PIACRA
1	1	PIACRB

Notice that the output registers are now in adjacent memory locations. This configuration can be used to advantage in applications where 16 bits must be brought into memory. With both the A and B sides established as input ports, the LDX and STX instructions can be used to efficiently transfer two bytes at a time. A specific example of this technique is described in Section 5-4. If this allocation is selected, initialization routines such as the first example of Section 3-4.1.3 can also be simplified:

```
10  LDX    #$2F24    ESTABLISH CONTROL MODES
20  STX    PIACRA    FOR BOTH SIDES.
```

In this sequence, the single instruction STX causes the appropriate constant to be loaded into both Control Registers.

[9]This assumes that b2 of the Control Registers has been set to select the Output Registers.

3-4.2 MC6850 ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER

3-4.2.1 Input/Output Configuration

The MC6850 Asynchronous Communications Interface Adapter (ACIA) provides a means of efficiently interfacing the MPU to devices requiring an asynchronous serial data format. The ACIA includes features for formatting and controlling such peripherals as Modems, CRT Terminals, and teletype printer/readers. An Input/Output Diagram of the MC6850 is shown in Figure 3-4.2.1-1.

Data flow between the MPU and the ACIA is via 8 bi-directional lines, DB0 through DB7, that interface with the MPU Data Bus. The direction of data flow is controlled by the MPU via the Read/Write input to the ACIA.

The "MPU side" of the ACIA also includes (see Figure 3-4.1.3-2) three chip select lines, CS0, CS1, and $\overline{CS2}$, for addressing a particular ACIA. An additional addressing input, Register Select (RS), is used to select specific registers within the ACIA. The MPU can read or write into the internal registers by addressing the ACIA via the system Address Bus using these four input lines. From the MPU's addressing point of view, each ACIA is simply two memory locations that are treated in the same manner as any other read/write memory.

The MPU also provides a timing signal to the ACIA via the Enable input. The Enable (E) pulse is used to condition the ACIA's internal interrupt control circuitry and for the timing of status/control changes. Since all data transfers take place during the $\phi 2$ portion of the clock cycle, $\phi 2$ is applied as the E signal.

The "Peripheral side" of the ACIA includes two serial data lines and three control lines. Data is transmitted and received via the Tx Data output and Rx Data inputs, respectively. Control signals Clear-To-Send ($\overline{CTS}$), Data Carrier Detect ($\overline{DCD}$), and Request-To-Send ($\overline{RTS}$) are provided for interfacing with Modems such as the MC6860. Two clock inputs are available for supplying individual data clock rates to the receiver and transmitter portions of the ACIA.

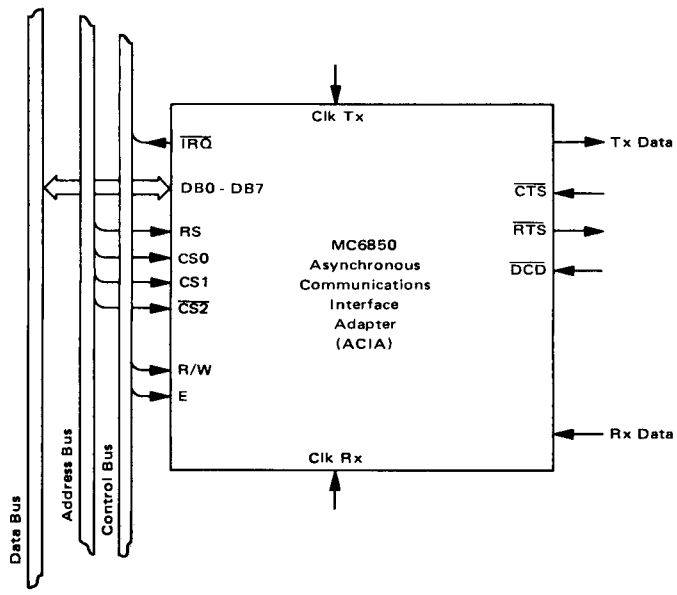


FIGURE 3-4.2.1-1: MC6850 ACIA I/O Diagram

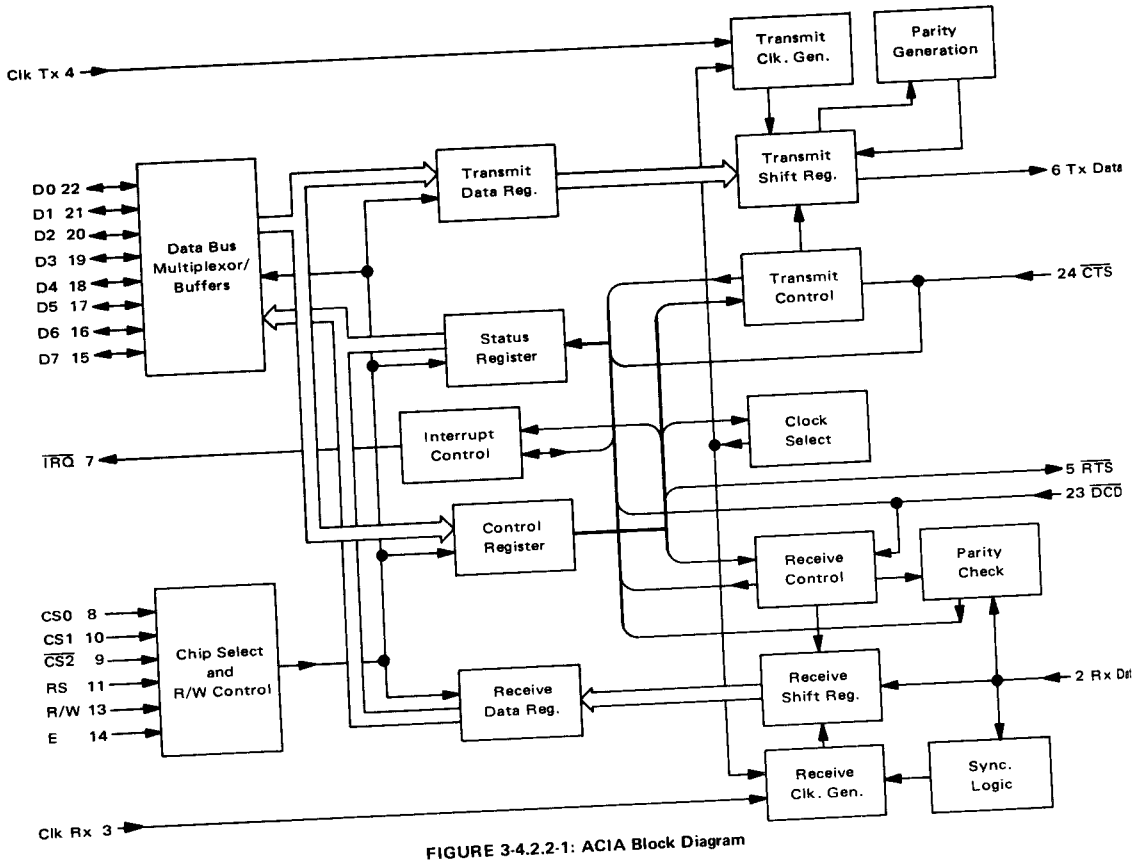


FIGURE 3-4.2.2-1: ACIA Block Diagram

3-4.2.2 Internal Organization

An expanded Block Diagram of the ACIA is shown in Figure 3-4.2.2-1. While the ACIA appears to the MPU as two addressable memory locations, internally there are four registers, two that are Write Only and two that are Read Only. The Read Only registers are for status and received data and the Write Only registers are for ACIA control and transmit data.

The Status Register format and a summary of the status bits is shown in Figure 3-4.2.2-2. The first two bits b0 and b1 indicate whether the Receiver Data Register is full (RDRF) or if the Transmit Data Register is empty (TDRE). b0 will go high when Rx data has been transferred to the Receiver Data Register (RDR). b0 will go low on the trailing edge of the Read Data command (reading the Receiver Data Buffer) or by a master reset command from bits b0 and b1 of the Control Register.

Status bit b1 (Tx Data Register Empty) will go high when a transmitter data transfer has taken place indicating that the Transmit Data Register (TDR) is available for new data entry from the MPU Bus. Bit b1 will return low on the trailing edge of a write data command. b1 will be held low if Clear-To-Send is not received from a peripheral device ($\overline{CTS}$ = "1")

Status bits b2 (Data Carrier Detect) and b3 (Clear-To-Send) are flag indicators from an external modem. Bit b2 ($\overline{DCD}$) will be high when the received carrier at the modem has been lost (ACIA's $\overline{DCD}$ input is high). Bit b2 will remain high until the interrupt is cleared by reading the Status Register and the Receiver Data Register. Bit b3 ($\overline{CTS}$) is low during reception of a Clear-To-Send command from a modem or other peripheral device.

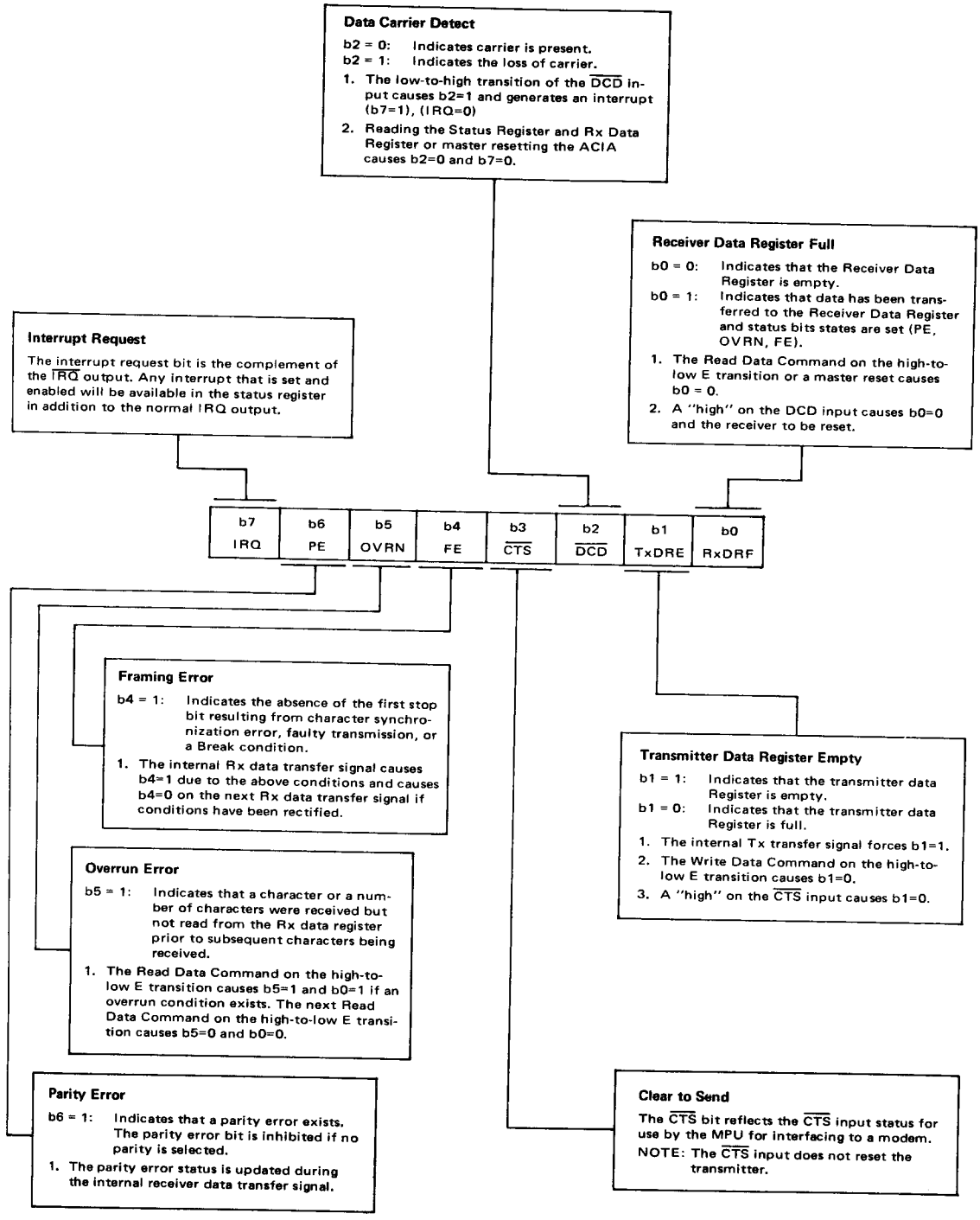


FIGURE 3-4.2.2-3: ACIA Status Register Format

Bit b4 (Framing Error) will be high whenever a data character is received with an improper start/stop bit character frame. The framing error flag b4 is cleared by the next data transfer signal if the condition causing the framing error has been rectified. Bit b5 (Receiver Overrun) being high indicates that the Receiver Data Register has not been read prior to a new character being received by the ACIA. This bit is cleared by reading the Receiver Data Register. Status Register bit b6 (Parity Error) is set whenever the number of high (''1's'') in the received character does not agree with the preselected odd or even parity. Bit b7 (Interrupt Request) when high indicates the ACIA is requesting interrupt to the MPU via the ACIA $\overline{IRQ}$ output and may be caused by b0 or b1 or b2 being set. All of the Status Register bits (except b3) will be cleared by an ACIA Master Reset.

The Control Register is an eight bit write only buffer which controls operation of the ACIA receiver, transmitter, interrupt enables, and the modem Request-To-Send control line. The Control Register format and a summary of its features is shown in Figure 3-4.2.2-3.

Control bits b0 and b1 select a Master Reset function for the ACIA when both bits are high and selects different clock divide ratios for the transmitter and receiver sections for the other combinations:

b1 (CDS2)	**b0** (CDS1)	**Clock Division**
0	0	÷ 1
0	1	÷16
1	0	÷64
1	1	Master Reset

The next 3 control bits, b2, b3, and b4, are provided for character length, parity, and stop bit selection. The encoding format is as follows:

b4 (WS3)	**b3** (WS2)	**b2** (WS1)	**Character Frame**
0	0	0	7 Bit + Even Parity + 2 Stop Bits
0	0	1	7 Bit + Odd Parity + 2 Stop Bits
0	1	0	7 Bit + Even Parity + 1 Stop Bit
0	1	1	7 Bit + Odd Parity + 1 Stop Bit
1	0	0	8 Bit + No Parity + 2 Stop Bits
1	0	1	8 Bit + No Parity + 1 Stop Bit
1	1	0	8 Bit + Even Parity + 1 Stop Bit
1	1	1	8 Bit + Odd Parity + 1 Stop Bit

The ACIA transmitter section is controlled by control bits b5 (TC1) and b6 (TC2). The four combinations of these two inputs provide transmission of a break command, Modem Request-To-Send ($\overline{RTS}$) command, and a transmitter inhibit/enable for the ACIA Interrupt Request output. When both b5 and b6 are low, the Request-To-Send ($\overline{RTS}$) output will be active low and the transmitter data register empty flag is enabled to the ACIA's Interrupt Request ($\overline{IRQ}$) output. If b5 is high and b6 is low the $\overline{RTS}$ output remains active low but the transmit IRQ input is inhibited. To turn off the $\overline{RTS}$ output b6 should be high and b5 low. This selection also enables the transmitter interrupt input to the $\overline{IRQ}$ output. When both b5 and b6 of the control register are high, Request-To-Send is on ($\overline{RTS}$) = 0, $\overline{IRQ}$ is enabled for the transmitter, and a break is transmitted (a space).

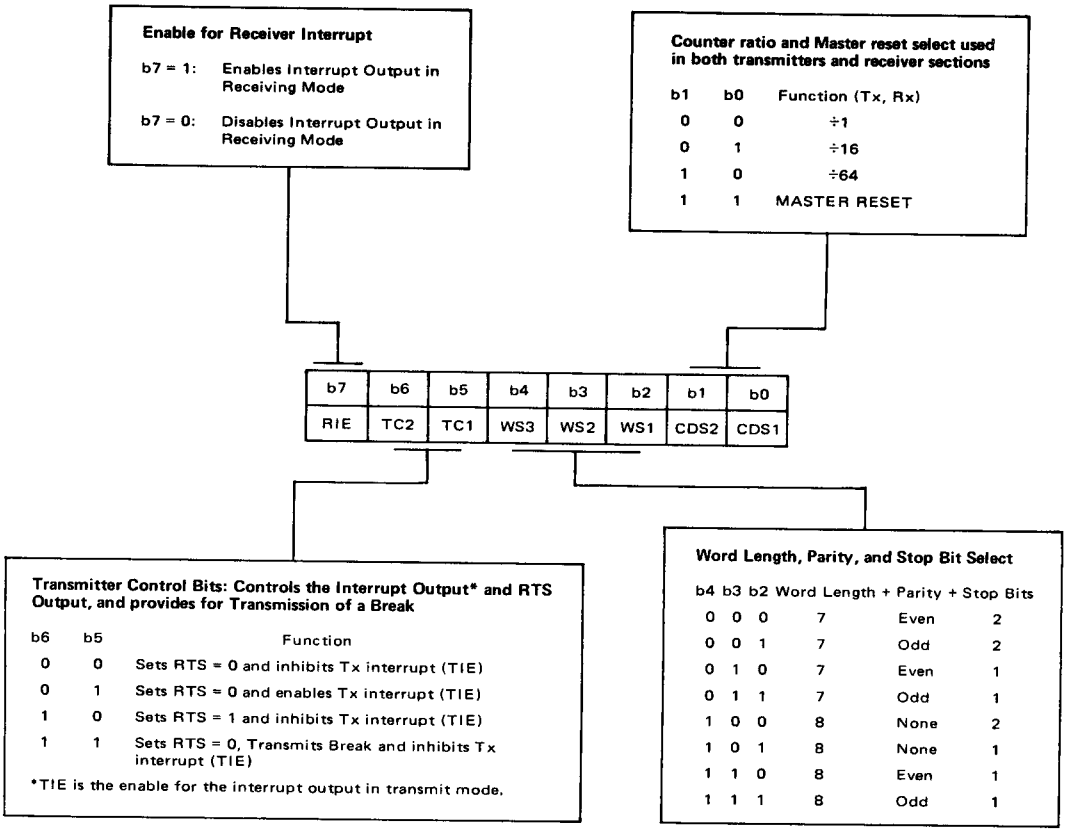


FIGURE 3-4.2.3-4: ACIA Control Register Format

Bits b7 controls the Receiver Interrupt Enable to the $\overline{IRQ}$ output. When b7 is high $\overline{IRQ}$ will indicate an interrupt request of the Receiver Data Register is Full (RDRF).

3-4.2.3 Addressing and Initialization

A specific example of ACIA usage is shown by the application described in Section 5-3, however, some basic considerations are discussed in the following paragraphs. As indicated in Section 3-4.1.2, the MPU addresses the ACIA via the chip select and register select inputs from the Address Bus. The correspondence between internal registers and the address inputs is shown in Figure 3-4.2.3-1.

With the chip selects properly enabled and RS = 0, either the Status or Control Register will be selected, depending on the current state of the Read/Write line: R/W = 0 = Write, Control Register is selected;

$\overline{CS2}$	CS1	CSϕ	RS	R/W	
ϕ	1	1	ϕ	ϕ	Control Register
ϕ	1	1	ϕ	1	Status Register
ϕ	1	1	1	ϕ	Transmit Data Register
ϕ	1	1	1	1	Receive Data Register
X	X	ϕ	X	X	ACIA Not Selected
X	ϕ	X	X	X	ACIA Not Selected
1	X	X	X	X	ACIA Not Selected

X = Don't Care

FIGURE 3-4.2.3-1: ACIA Register Addressing

R/W = 1 = Read, Status Register is selected. Similarly, when RS = 1, either the Receive Data Register (R/W = 1 = Read) or the Transmit Data Register (R/W = 0 = Write) is selected.

Addressing the ACIA can be illustrated in conjunction with the simple system configuration shown in Figure 3-4.1.3-2[10]. The method shown is typical for assigning mutually exclusive memory addresses to the family devices without the use of additional decode logic. The connections shown assign memory addresses as follows:

RAM	0000 – 007F
PIA	4004 – 4007
ACIA	4008 – 4009
ROM	C000 – C3FF

(Hexadecimal notation)

As voltage is applied to the ACIA during the power-on sequence, its internal registers are cleared to zero[11] by circuitry within the ACIA to prevent spurious outputs. This initial condition means that interrupts are disabled, IRQ to the MPU is high (no interrupt request), and the Ready-To-Send, $\overline{RTS}$, output is high. The first step in preparation for using the ACIA must be a master reset via bits b0 and b1 of the Control Register, that is, the MPU must write ones into those positions. Once reset, the ACIA operating mode is established by writing the appropriate data into the Control Register.

3-4.2.4 System Considerations

The ACIA is used primarily to transfer serial data between the microprocessor and real time peripheral devices such as teletypes, CRT terminals, etc. The most common data format used for the transfer of real-time data is the asynchronous data format. Use of this format is generally limited to low transmission rates — below 1200 bps or 120 char/sec. For example, the maximum transmission rate of a teletype is 10 char/sec. Here, the transmission of data to the MPU depends on the operator's dexterity of depressing a key on the keyboards. Since the transmission of data is dependent on the operator, gaps (non transmission of data) between data characters occur as a general rule.

In the transmission of asynchronous data, there is no pre-synchronized clock provided along with the data. Also, the gaps between data characters in this transmission mode requires that synchronization be re-established for each character. Therefore, the receiving device must be capable of establishing bit and

[10] Figure 3-4.1.3-1 is identical to Figure 1-1.2-1 and is discussed in Section 1-1.2 of Chapter 1.

[11] If external high signals are present on the $\overline{DCD}$ and $\overline{CTS}$ inputs, their respective bits, b2 and b3, in the Status Register will also be high.

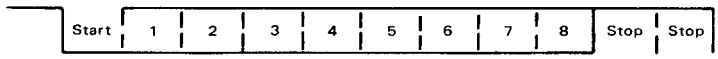
character synchronization from the characteristics of the asynchronous format. Each character consists of a specified number of data bits preceded by a start bit and followed by one or more stop bits as shown in Figure 3-4.2.4-1.

These start and stop elements do not contain any information and they actually slow down the effective transmission rate. Since the asynchronous format is used in real time systems, the effect of the start and stop bits on the transmission rate is negligible. The purpose of the start bit is to enable a receiving system to synchronize its clock to this bit for sampling purposes and thereby establish character synchronization. The stop bit is used as a final check on the character synchronization.

Since the MPU processes eight bit parallel bytes that do not include start and stop elements, received serial data in an asynchronous format must be converted to parallel form with the start and stop elements stripped from the character. Likewise, in order to transmit serial data the parallel data byte from the MPU must be converted to serial form with the start and stop elements added to the character. This serial-to-serial/parallel-to-parallel conversion is the primary function of the ACIA.

Desired options such as variable clock divider ratios, variable word length, one or two stop bits, odd or even parity, etc. are established by writing an appropriate constant into the ACIA's Control Register. The combination of options selected depends on the desired format for a particular application. The general characteristics of data flow through the ACIA are described in the following paragraphs.

A typical transmitting sequence consists of reading the ACIA status register either as a result of an interrupt or in the ACIA's turn in a polling sequence. A character may be written into the Transmit Data Register if the status read operation has indicated that the Transmit Data Register is empty. This character is transferred to a shift register where it is serialized and transmitted from the Tx Data output preceded by a start bit and followed by one or two stop bits. Internal parity (odd or even) can be optionally added to the character and will occur between the last data bit and the first stop bit. After the first character is written in the data register, the Status Register can be read again to check for a Transmit Data Register Empty condition and current peripheral status. If the register is empty, another character can be loaded for transmission even though the first character is in the process of being transmitted. This second character will be automatically transferred into the shift register when the first character transmission is completed. The above sequence may be continued until all the characters have been transmitted.



Start Bit – "Space" – Logic Zero
Start Bits – "Mark" – Logic One
Idling Bits – "Mark"

FIGURE 3-4.2.4-1: Asynchronous Data Format

Data is received from a peripheral by means of the Rx Data input. A divide by one clock ratio is provided for an external clock that is synchronized to its data; the divide by 16 and 64 ratios may be used for internal synchronization. Bit synchronization in the divide by 16 and 64 modes is obtained by detecting the leading mark-to-space transition of the start bit. False start bit detection capability insures that a full half bit of a start bit has been received before the internal clock is synchronized to the bit time. As a character is being received, parity (odd or even) will be checked and the possible error indication will be available in the status register along with framing error, overrun error, and receiver data register full. In a typical receiving sequence, the Status Register is read to determine if a character has been received from a peripheral. If the receiver data register is full, the character is placed on the Data Bus when the MPU reads the ACIA Receive Data Register. The status register can be read again to determine if another character is available in the receiver data register. The receiver is also double buffered so that a character can be read from the data register as another character is being received in the shift register. The above sequence may be continued until all characters have been received.

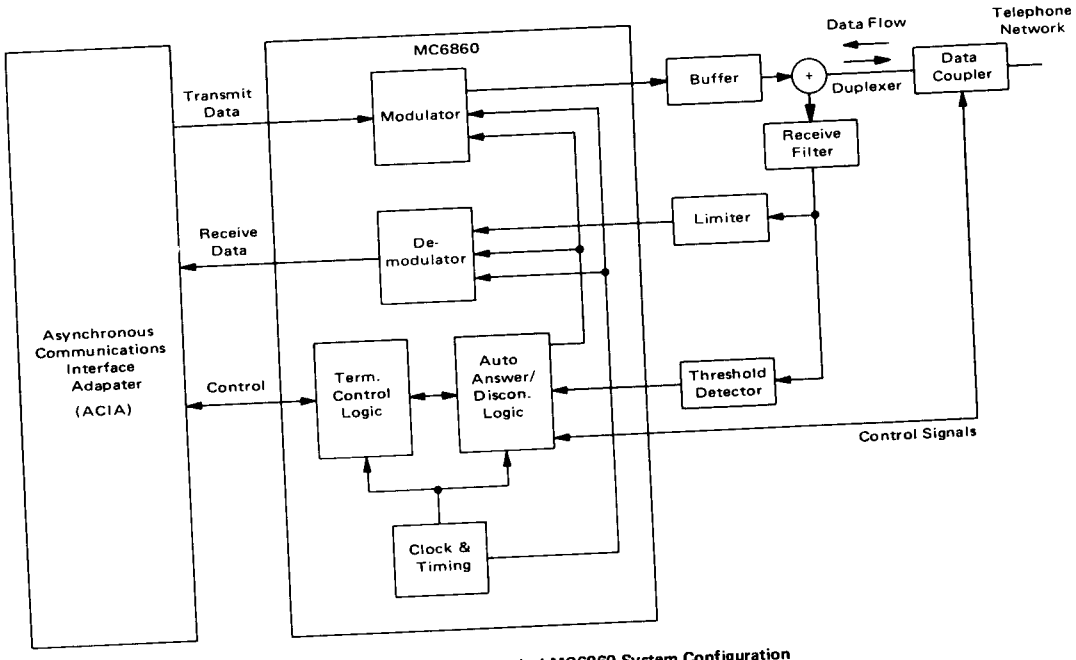


FIGURE 3-4.3.1-1: Typical MC6860 System Configuration

3-4.3 MC6860 LOW SPEED MODEM

3-4.3.1 Input/Output Configuration

The MC6860 Modem provides a very effective method of interfacing a MPU based system, via a MC6850 ACIA, to a telephone network as shown in Figure 3-4.3.1-1. The modem provides full automatic answer/originate and initiate disconnect capability under MPU program control thru the ACIA. Data may be asynchronously sent and received over the telephone network at data rates up to 600 bits per second.

The Input/Output configuration of the MC6860 when used with the MC6850 ACIA and the MC6800 MPU family is shown in Figure 3-4.3.1-2. Data flow from the terminal side of the modem enters in serial digital format via the transmit data line of the modem. It is then digitally processed by the modulator section and exits the telephone network side of the modem via the transmit carrier line. This digitized sinewave FSK signal is post filtered by an output buffer/low pass filter. The filtered analog sinewave passes through a line duplexer to the telephone line via a data coupler.

The returning analog signal from the remote modem at the other end of the telephone line passes through the data coupler and duplexer and is applied to a bandpass filter/amplifier. The receive bandpass filter bandlimits the incoming signal to remove noise and adjacent transmit channel interference. After being band-limited the analog signal is full limited to a 50% duty cycle TTL level signal by the input limiter. This digital signal is the receive carrier that is applied to the modem. The output signal from the bandpass filter is also routed to a threshold detector to determine if the input signal to the limiter is above the minimum detectable signal level presented to the modem. When the signal input level exceeds the bias point of the threshold detector, the detector's output goes low at the threshold input pin to the MC6860 modem indicating that carrier is present.

A complete listing and functional description of all I/O pins for the MC6860 (Figure3-4.3.2-1) is provided in the following:

Data Terminal Ready ($\overline{DTR}$)

The Data Terminal Ready signal must be low before the modem function will be enabled. To initiate a disconnect, $\overline{DTR}$ is held high for 34 msec minimum. A disconnect will occur 3 seconds later.

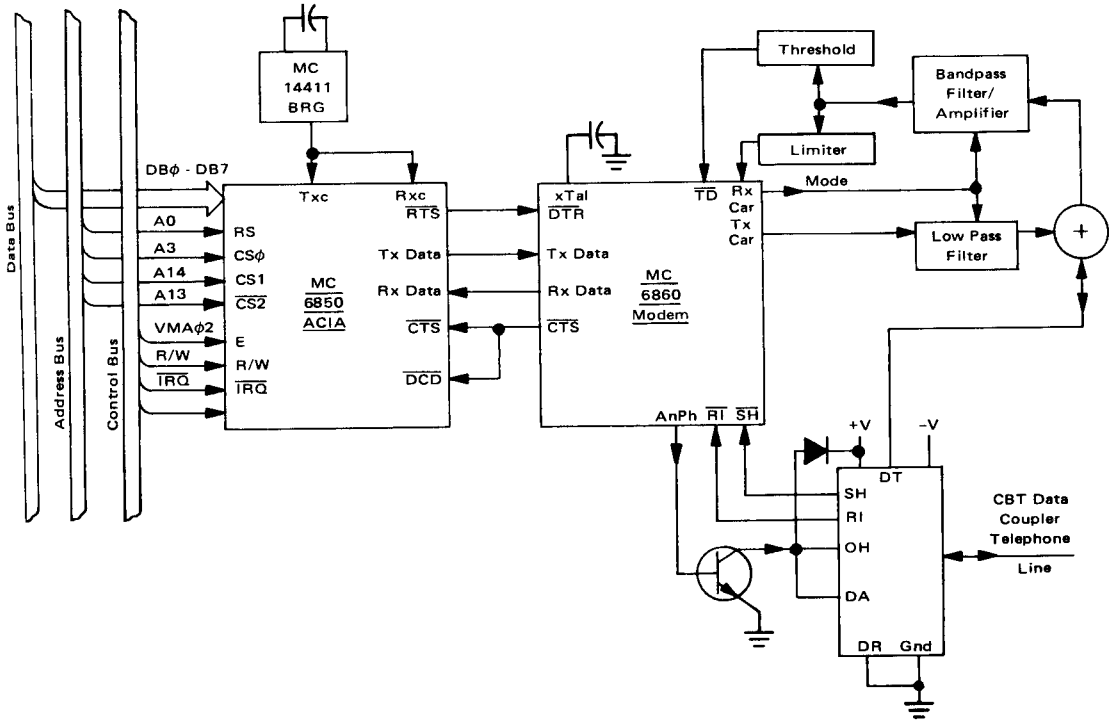


FIGURE 3-4.3.1-2: I/O Configuration For MC6860 Modem

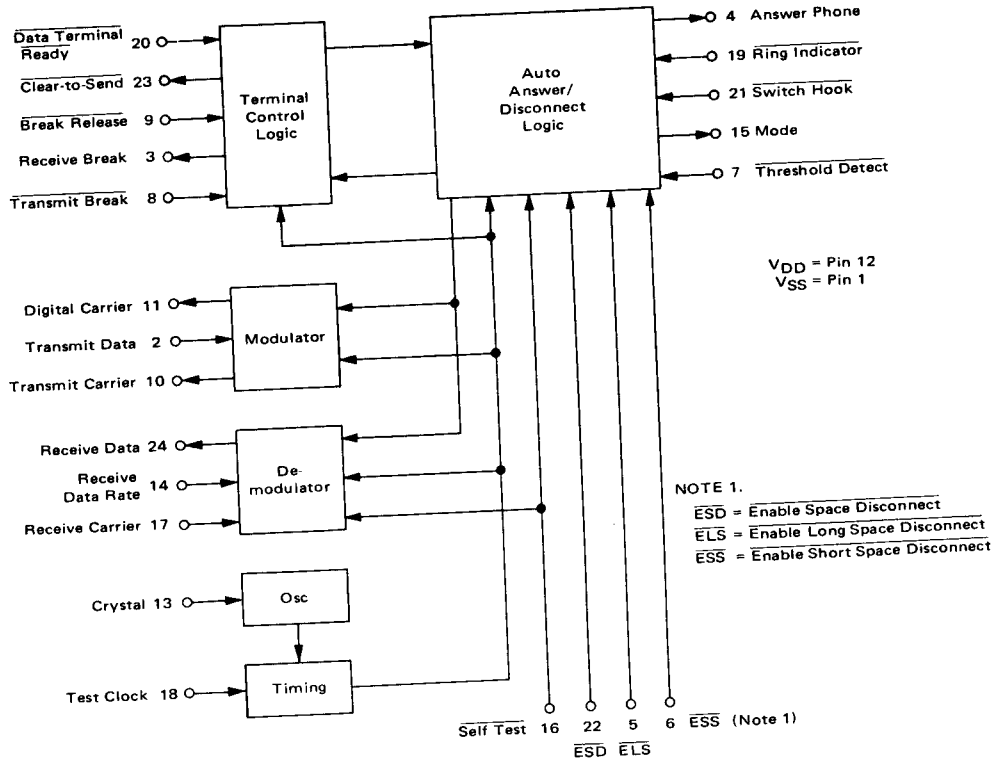


FIGURE 3-4.3-2-1: MC6860 Modem Block Diagram

Clear-To-Send ($\overline{CTS}$)

A low on the $\overline{CTS}$ output indicates the Transmit Data input has been unclamped from a steady Mark, thus allowing data transmission.

Ring Indicator ($\overline{RI}$)

The modem function will recognize a receipt of a call from the CBT if at least 20 cycles of the 20-47 Hz ringing signal are present. The CBS $\overline{RI}$ signal must be level-converted from EIA RS-232 levels before interfacing it with the modem function. The receipt of a call from the CBS is recognized if the $\overline{RI}$ signal is present for at least 51 msec. This input is held high except during ringing. A $\overline{RI}$ signal automatically places the modem function in the Answer Mode.

Switch Hook ($\overline{SH}$)

$\overline{SH}$ interfaces directly with the CBT and via a EIA RS-232 level conversion for the CBS. An $\overline{SH}$ signal automatically places the modem function in the Originate Mode.

$\overline{SH}$ is low during origination of a call. The modem will automatically hang up 17 seconds after the release of $\overline{SH}$ if the handshaking routine between the local and remote modem has not been accomplished.

Threshold Detect ($\overline{TD}$)

This input is derived from an external threshold detector. If the signal level is sufficient, the $\overline{TD}$ input must be low for 20μs at least once every 32 msec to maintain normal operation. An insufficient signal level indicates the absence of the Receive Carrier; an absence for greater than 32 msec will not cause channel establishment to be lost; however, data during this interval will be invalid.

Answer Phone (An Ph)

Upon receipt of Ring Indicator or Switch Hook signal and Data Terminal Ready, the Answer Phone output goes high [($\overline{SH}$ + $\overline{RI}$) • $\overline{DTR}$]. This signal drives the base of a transistor which activates the Off Hook (OH) and Data Transmission (DA) control lines in the data coupler. Upon call completion, the Answer Phone signal returns to a low level.

Mode

The Mode output indicates the Answer (low) or Originate (high) status of the modem. This output changes state when a Self Test command is applied.

Transmit Break ($\overline{Tx\ Brk}$)

The Break command is used to signal the remote modem to stop sending data.

A Transmit Break (low) greater than 34 msec forces the modem to send a continuous space signal for 233 msec. Transmit Break must be initiated only after $\overline{CTS}$ has been established. This is a negative edge sense input. Prior to initiating $\overline{Tx\ Brk}$, this input must be held high for a minimum of 34 msec.

Receive Break (Rx Brk)

Upon receipt of a continuous 150 msec space, the modem automatically clamps the Receive Break output high. This output is also clamped high until Clear-To-Send is established.

Break Release ($\overline{Brk\ R}$)

After receiving a 150 msec space signal, the clamped high condition of the Receive Break output can be removed by holding Break Release low for at least 20 μs.

Transmit Data (Tx Data)

Transmit Data is the binary information presented to the modem function for modulation with FSK techniques. A high level represents a Mark.

Receive Data (Rx Data)

The Receive Data output is the data resulting from demodulating the Receive Carrier. A Mark is a high level.

Receive Data Rate (Rx Rate)

The demodulator has been optimized for signal-to-noise performance at 300 bps and 600 bps. The Receive Data Rate input should be low for 0-600 bps and should be high for 0-300 bps.

Digital Carrier (FO)

A test signal output is provided to decrease the chip test time. The signal is a square wave at the transmit frequency.

Transmit Carrier (Tx Car)

The Transmit Carrier is a digitally-synthesized sinewave derived from the 1.0 MHz crystal reference. The frequency characteristics are as follows:

Mode	Data	Transmit Frequency	Accuracy*
Originate	Mark	1270 Hz	−0.15 HZ
Originate	Space	1070 Hz	+0.09 Hz
Answer	Mark	2225 Hz	−0.31 Hz
Answer	Space	2025 Hz	−0.71 Hz

*The reference frequency tolerance is not included.

The proper output frequency is transmitted within the 3.0 μs following a data bit change with no more than 2.0 μs phase discontinuity. The typical output level is 0.35 V (RMS) into a 200 k-ohm load impedance.

The second harmonic is typically 32 dB below the fundamental.

Receive Carrier (Rx Car)

The Receive Carrier is the FSK input to the demodulator. The local Transmit Carrier must be balanced or filtered out prior to this input, leaving only the Receive Carrier in the signal. The Receive Carrier must also be hard limited. Any half-cycle period greater than or equal to 429 ± 1.0 μs for the low band or 235 ± 1.0 μs for the high band is detected as a space.

Enabled Space Disconnect ($\overline{\text{ESD}}$)

When $\overline{\text{ESD}}$ is strapped low and $\overline{\text{DTR}}$ is pulsed to initiate a disconnect, the modem transmits a space for either 3 seconds or until a loss of threshold is detected, whichever occurs first. If $\overline{\text{ESD}}$ is strapped high, data instead of a space is transmitted. A disconnect occurs at the end of 3 seconds.

Enable Short Space Disconnect ($\overline{\text{ESS}}$)

$\overline{\text{ESS}}$ is a strapping option which, when low, will automatically hang up the phone upon receipt of a continuous space for 0.3 seconds. *$\overline{\text{ESS}}$ and $\overline{\text{ELS}}$ must not be simultaneously strapped low.*

Enable Long Space Disconnect ($\overline{\text{ELS}}$)

ELS is a strapping option which, when low, will automatically hang up the phone upon receipt of a continuous space for 1.5 seconds.

Crystal (Xtal)

A 1.0-MHz crystal with the following parameters is required to utilize the on-chip oscillator. A 1.0-MHz square wave can also be fed into this input to satisfy the clock requirement.

Mode:	Parallel
Frequency:	1.0 MHz ±0.1%
Series Resistance:	750 ohms max
Shunt Capacitance:	7.0 pF max
Temperature:	0-70°C
Test Level:	1.0 mW
Load Capacitance:	13 pF

When utilizing the 1.0-MHz crystal, external parasitic capacitance, including crystal shunt capacitance, must be ≤9 pF at the crystal input.

Test Clock ($\overline{TST}$)

A test signal input is provided to decrease the test time of the chip. In normal operation this input *must be strapped low.*

Self Test ($\overline{ST}$)

When a low voltage level is placed on this input, the demodulator is switched to the modulator frequency and demodulates the transmitted FSK signal. Channel establishment, which occurred during the initial handshake, is not lost during self test. The Mode Control output changes state during $\overline{\text{Self Test}}$, permitting the receive filters to pass the local Transmit Carrier.

INPUTS			OUTPUT
ST	SH	RI	Mode
H	L	H	H
H	H	L	L
L	L	H	L
L	H	L	H

MODE CONTROL TRUTH TABLE

3-4.3.2 Internal Organization

The MC6860 Modem may be broken down into internal functional sections as shown in Figure 3-4.3.2-1. The terminal control logic and auto answer/disconnect logic sections are referred to as the supervisory control section. This section contains digital counters which provide the required time out intervals and necessary control gating logic. This provides logic outputs Clear-To-Send and Answer Phone from inputs Ring Indicator, Switch Hook, and Data Terminal Ready. Also the control section has some local strapping options available on pins 5, 6, and 22. These options provide time outs for line hang-up or termination of the data communication channel.

The oscillator/timing blocks accept a 1.0 MHz clock into pin 13 either from an external clock source or by connecting a 1.0 MHz crystal between pin 13 and ground. A test clock input is provided to allow more rapid testing of the MC6860 timing chains used for various timeouts. This input must be strapped low during normal operation.

The modulator section takes the input digital data and converts it to one of two FSK tones for transmission over the telephone network. There are two tones for transmission and two tones used for reception during full depulx operation. During data transmission from the call origination modem the transmit tones are: 1270 Hz for a Mark and 1070 Hz for a Space. This originating modem will receive two frequencies in the high band which are: 2225 Hz for a Mark and 2025 Hz for a space. If the local modem answers the data call it will transmit in the high band 2225/2025 Hz and receive in the low band 1270/1070 Hz. The modulator section generates these frequencies digitally by synthesizing a sinewave with an 8 step D to A available on pin 10 and a digital square wave output at the above frequencies available on pin 11.

The demodulator accepts a 50% duty cycle TTL level square wave derived from amplifying, filtering, and limiting the incoming line FSK analog signal. The binary data is recovered from the FSK signal by detecting when the signal has a zero crossing and digitally using post detection techniques to discriminate

between the two incoming mark/space tones. A receive data rate input (pin 14) is used to optimize the post detection filter at either 300 or 600 bits per second.

3-4.3.3 Handshaking and Control

The supervisory control section of the modem can function in four different modes. Two are associated with data communication channel initialization (Answer Mode and Originate Mode) and two are for channel termination or hang-up (Automatic Disconnect and Initiate Disconnect).

Answer Mode

Automatic answering is first initiated by a receipt of a Ring Indicator ($\overline{RI}$) signal. This can be either a low level for at least 51 msec as would come from a CBS data coupler, or at least 20 cycles of a 20-47 Hz ringing signal as would come from a CBT data coupler. The presence of the Ring Indicator signal places the modem in the Answer Mode; if the Data Terminal Ready line is low, indicating the communication terminal is ready to send or receive data, the Answer Phone output goes high. This output is designed to drive a transistor switch which will activate the Off Hook (OH) and Data Transmission (DA) relays in the data coupler. Upon answering the phone the 2225-Hz transmit carrier is turned on.

The originate modem at the other end detects this 2225-Hz signal and after a 450 msec delay (used to disable any echo suppressors in the telephone network) transmits a 1270-Hz signal which the local answering modem detects provided the amplitude and frequency requirements are met. The amplitude threshold is set external to the modem chip. If the signal level is sufficient the $\overline{TD}$ input should be low for 20 μs at least once every 32 msec. The absence of a threshold indication for a period greater than 51 msec denotes the loss of Receive Carrier and the modem begins hang-up procedures. Hang-up will occur 17 seconds after $\overline{RI}$ has been released provided the handshaking routine is not re-established. The frequency tolerance during handshaking is $\pm$100 Hz from the Mark frequency.

After the 1270-Hz signal has been received for 150 msec, the Receive Data is unclamped from a Mark condition and data can be received. The Clear-To-Send output goes low 450 msec after the receipt of carrier and data presented to the answer modem is transmitted.

Automatic Disconnect

Upon receipt of a space of 150 msec or greater duration, the modem clamps the Receive Break high. This condition exists until a Break Release command is issued at the receiving station. Upon receipt of a 0.3 second space, with Enable Short Space Disconnect at the most negative voltage (low), the modem automatically hangs up. If Enable Long Space Disconnect is low, the modem requires 1.5 seconds of continuous space to hang up.

Originate Mode

Upon receipt of a Switch Hook ($\overline{SH}$) command the modem function is placed in the Originate Mode. If the Data Terminal Ready input is enabled (low) the modem will provide a logic high output at Answer Phone. The modem is now ready to receive the 2225-Hz signal from the remote answering modem. It will continue to look for this signal until 17 seconds after SH has been released. Disconnect occurs if the handshaking routine is not established.

Upon receiving 2225 $\pm$100 Hz for 150 msec at an acceptable amplitude, the Receive Data output is unclamped from a Mark condition and data reception can be accomplished. 450 msec after receiving a 2225-Hz

signal, a 1270-Hz signal is transmitted to the remote modem. 750 msec after receiving the 2225-Hz signal, the Clear-To-Send output is taken low and data can now be transmitted as well as received.

Initiate Disconnect

In order to command the remote modem to automatically hang up, a disconnect signal is sent by the local modem. This is accomplished by pulsing the normally low Data Terminal Ready into a high state for greater than 34 msec. The local modem then sends a 3 second continuous space and hangs up provided the Enable Space Disconnect is low. If the remote modem hangs up before 3 seconds, loss of Threshold Detect will cause loss of Clear-To-Send, which marks the line in Answer Mode and turns the carrier off in the Originate Mode.

If $\overline{\text{ESD}}$ is high the modem will transmit data until hang-up occurs 3 seconds later. Transmit Break is clamped 150 msec following the Data Terminal Ready interrupt.

Each of the four above operational modes are shown in Figures 3-4.3.3-1 through 3-4.3.3-4.

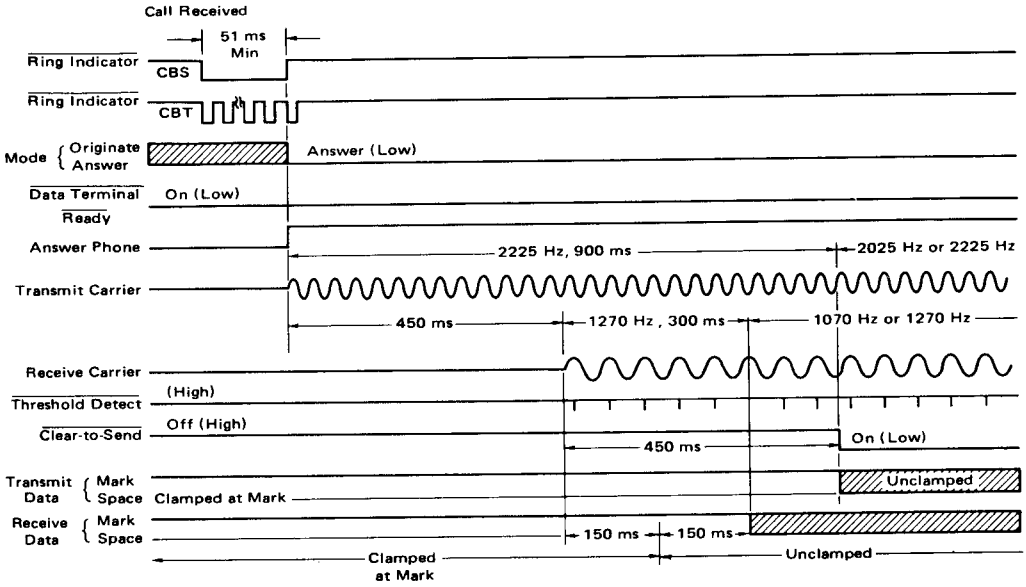


FIGURE 3-4.3.3-1: Answer Mode

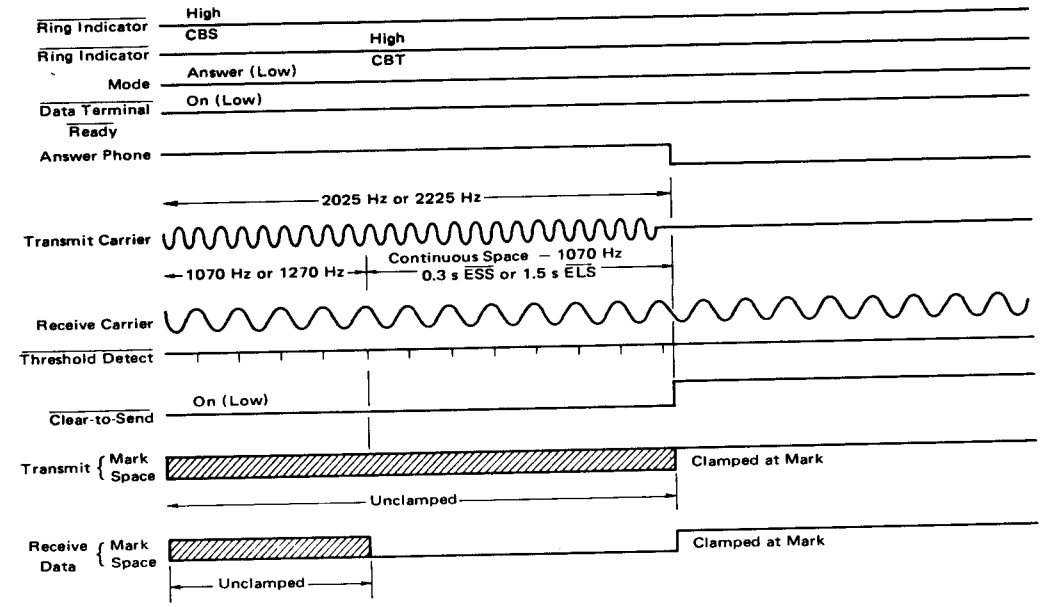


FIGURE 3-4.3.3-2: Automatic Disconnect - Long or Short Space

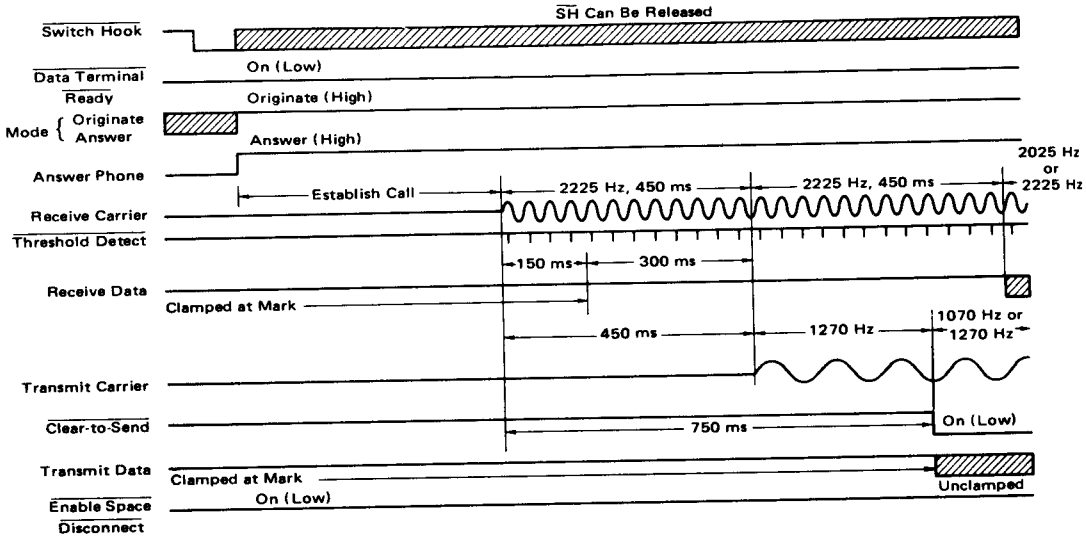


FIGURE 3-4.3.3-3: Originate Mode

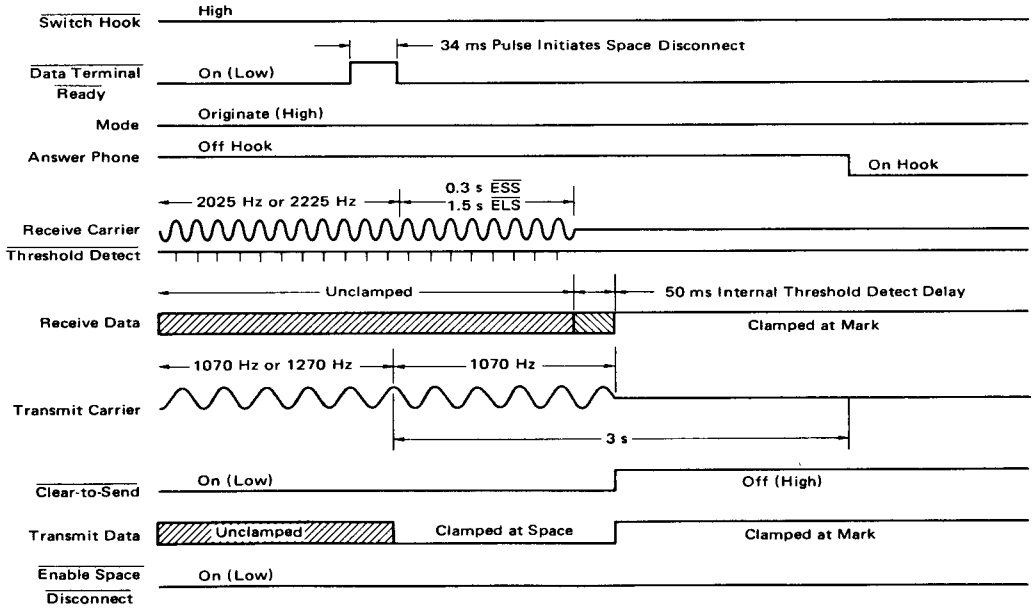


FIGURE 3-4.3.3-4: Initiate Disconnect

3-5 DIRECT MEMORY ACCESS

The term Direct Memory Access (DMA) is applied to a variety of techniques for speeding up overall system operation by loading and unloading memory faster than can be done using an MPU control program. DMA is often described as a means of allowing fast peripherals (perhaps another Microprocessor), to access the system memory without "bothering" the MPU. However, most DMA procedures do interfere with normal operation to some extent. The capability for handling the various techniques is an often used figure of merit for evaluating Microprocessors.

The MC6800's supervisory control features permit any of three commonly used DMA techniques to be used; (1) Transfer data with MPU halted; (2) Transfer data on burst basis (cycle stealing) with MPU running; (3) Transfer data synchronously with MPU running. Methods for implementing each of these techniques are described in Section 4-2.2 therefore, only qualitative descriptions are included here.

The simplest procedure for DMA merely uses the $\overline{\text{Halt}}$ control to shut the MPU down while the DMA takes place. In the Halt state, the MC6800 effectively removes itself from the Address and Data Buses by putting all buffers in the high impedence off state. This method has the disadvantage that it can take a relatively long time for the MPU to "vacate" the buses. The MC6800 is designed to finish executing its current instruction before entering the Halt or Wait state; the resulting delay depends on which instruction is being executed and may be as much as 13 machine (clock) cycles. However, due to its simplicity this is the preferred method if the delay can be tolerated and long transfers are required.

In contrast to this, the Three-State Control (TSC) may be used to obtain DMA control within 500 nanoseconds of initiation but must be used only for short transfers. Activation of TSC puts the MPU's buffers in the high impedence off state. This technique has the disadvantage that activation of TSC should be synchronized with the $\phi 1$ clock and both clocks must be "frozen" ($\phi 1$ high, $\phi 2$ low) for the duration of the DMA. Due to the MPU's address and R/W refresh requirements, the clocks can only be frozen for a maximum of 5 microseconds, thus limiting the duration of the transfer.

A third method can be used that is completely transparent to the MPU. This technique takes advantage of the fact that MPU data transfers take place only during $\phi 2$ of the clock cycle. If the DMA control signals are properly synchronized and the memory is fast enough, DMA can be accomplished during $\phi 1$ of each clock cycle.

Each of these three methods is described in greater detail in Section 4-2.2. It should be noted that the faster methods impose additional external hardware requirements on the system.

The techniques described above of course do not exhaust all methods for performing DMA. As an additional example, DMA can be program controlled in the sense that a control program and hence the MPU can be used to establish the memory area to be used and to grant permission for the DMA. In this case the DMA circuitry is treated as another peripheral from which status and control signals can be passed through a PIA. This technique is also outlined in Section 4-2.2.